UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

GENERACIÓN DE CONJUNTOS DE MÁQUINAS DE SOPORTE VECTORIAL MEDIANTE TÉCNICAS DE REMUESTREO E INYECCIÓN DE RUIDO EN LAS ETIQUETAS DE CLASE

Autor: Luis Salamanca Polo
Tutor: Alberto Suárez González

Mayo 2017

GENERACIÓN DE CONJUNTOS DE MÁQUINAS DE SOPORTE VECTORIAL MEDIANTE TÉCNICAS DE REMUESTREO E INYECCIÓN DE RUIDO EN LAS ETIQUETAS DE CLASE

AUTOR: Luis Salamanca Polo
TUTOR: Alberto Suárez González

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Mayo de 2017

Resumen

Este trabajo tiene como objetivo mejorar los sistemas de aprendizaje automático basados en la generación de conjuntos de predictores. Como algoritmo base se utilizarán las máquinas de soporte vectorial. Dentro de problemas del aprendizaje supervisado el trabajo aborda los problemas de clasificación binaria.

Para desarrollar el objetivo enunciado, se ha extendido la funcionalidad de la librería EnsembleSVM. En esta librería la diversidad en el conjunto se genera utilizando técnicas de remuestreo, como *bagging* y *subbagging*. La extensión realizada consiste en incorporar un mecanismo adicional para generar diversidad entre los predictores del conjunto denominado *class-switching*. Esta técnica está basada en inducir predictores a partir de un conjunto de datos en los que se ha inyectado ruido en las etiquetas clase de manera aleatoria.

Una vez implementada esta funcionalidad adicional se lleva a cabo una investigación empírica de la eficacia, en términos de tasas de acierto y eficiencia, en términos de tiempo de entrenamiento, de los conjuntos generados. En el entrenamiento se aplican diferentes porcentajes de remuestreo y diferentes porcentajes de inyección de ruido en las etiquetas clase. En las pruebas realizadas, veremos que predecir con un conjunto de SVMs generalmente no mejora la tasa de error de la predicción de una única SVM. Este resultado confirma las observaciones realizadas en trabajos previos. En cambio, aplicando técnicas de submuestreo se pueden generar conjuntos con un tiempo de entrenamiento reducido, cuya capacidad de generalización es comparable a la de una única SVM entrenada con todos los ejemplos.

Palabras clave

Aprendizaje automático inductivo, clasificación supervisada, Máquinas de soporte vectorial, métodos de núcleo, LIBSVM, EnsembleSVM, métodos de conjuntos, *bagging*, *class-switching*.

Abstract

This work aims to improve the automatic learning systems based on the generation of sets of predictors. As base algorithm, support vector machines will be used. Within supervised learning problems, this essay approaches binary classification problems.

To achieve this objective, the functionality of the EnsembleSVM library has been extended. In this library, the diversity in the set is generated using resampling techniques, such as bagging and subbagging. The developed extension consists of incorporating an additional mechanism, in order to generate diversity among the predictors of the so-called "class-switching set". This technique is based on inducing predictors from a data set, with noise injected randomly into the class labels.

Once this additional functionality has been implemented, an empirical investigation of the efficiency is carried out. This is done regarding success rates and efficiency in one hand, and training time on the other. During training, different percentages of resampling and different percentages of noise injection in the classes are applied. During the test phase, a prediction within a set of SVMs will be proven not to improve the prediction error rate of a single SVM. This result confirms the observations done in previous works. In contrast, sets with a reduced training time can be generated applying subsampling techniques. The generalization capacity is comparable to a single trained SVM with all the examples.

Keywords

Induction machine learning, Supervised learning, Support Vector Machines, kernel methods, LIBSVM, EnsembleSVM, Ensemble, bagging, class-switching.

Agradecimientos

Quiero agradecer tanto a mis padres como a mi hermana todo el esfuerzo y el apoyo aportado a lo largo de mi carrera, principalmente por ayudarme a superar obstáculos en el camino.

También a la Universidad Autónoma de Madrid, en particular a mis profesores que me han transmitido su sabiduría y conocimientos y a mis compañeros que han hecho de mi apreciada universidad mi casa.

Por último y no por ello menos importante, a mi tutor, Alberto, por su paciencia, dedicación y ayuda que me ha prestado en todo momento sin importarle el tiempo. Ha sido muy importante su ayuda para llegar hasta aquí.

A todos ellos muchísimas gracias.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción y Objetivos

Este trabajo se enmarca dentro la Inteligencia Artificial (AI). El objetivo de esta área de la Ingeniería Informática es el desarrollo de sistemas autónomos cuyo comportamiento pueda ser considerado inteligente. Dentro de la Inteligencia Artificial, se ha realizado una contribución en el campo del aprendizaje automático. La finalidad del aprendizaje automático, es el desarrollo de sistemas que, de manera autónoma, puedan modificar su comportamiento a partir de la interacción con el medio. En concreto, el trabajo se desarrolla en el campo del aprendizaje supervisado. El objetivo de este tipo de aprendizaje es predecir características de un ejemplo (las etiquetas de clase), a partir de otras características (los atributos), utilizando para ello un conjunto de ejemplos previamente etiquetados. El trabajo realizado ha consistido en ampliar los métodos de generación de conjuntos de máquinas de vectores soporte utilizando tanto remuestreo como inyección de ruido en las etiquetas de clase como mecanismos de diversificación. En concreto, se ha realizado una extensión de la librería EnsembleSVM (Claesen, 2014), mediante la cual se pueden generar conjuntos de SVMs basados en técnicas de remuestreo.

1.1 Introducción

La inteligencia artificial, también conocida por las siglas IA (*Artificial Intelligence*), es la rama de las ciencias de la computación, cuyo objetivo es desarrollar sistemas autónomos inteligentes. Dentro del ámbito científico el término Inteligencia Artificial, aparece por primera vez en el título de un proyecto de investigación a realizar en Dartmouth College, Hanover, New Hampshire por John McCarthy, Marvin Minsky, N. Rochester y Claude Shanon, en 1956 (Russell & Norvig, 2016). En el resumen que encabeza la propuesta se describe el proyecto de la Inteligencia Artificial de la siguiente manera:

"*The study is to proceed on the basis of the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves*" McCarthy (Russell & Norvig, 2016).

Dentro de la Inteligencia Artificial, se distinguen dos vertientes la fuerte (*strong*) y la débil (*weak*). La inteligencia artificial fuerte tiene como objetivo el desarrollo de una inteligencia real, de forma que el sistema inteligente posea habilidades cognitivas similares o superiores a las de los seres humanos (Indika, 2011). En cambio, en su enfoque débil la Inteligencia Artificial tiene como objetico el desarrollo de sistemas capaces de resolver problemas concretos. Para que el sistema pueda resolver estos problemas basta con simular un comportamiento inteligente en el ámbito del problema concreto (Indika, 2011). La gran mayoría de los proyectos actuales de Inteligencia Artificial, incluyendo el aprendizaje automático, se enmarcan dentro de la interpretación débil del proyecto.

El aprendizaje automático es una rama de la inteligencia artificial cuyo objetivo es el diseño de sistemas capaces de adaptarse, mejorando su comportamiento de manera autónoma, por interacción con el medio o un instructor. Dentro del aprendizaje automático existen diferentes tipos de problemas a resolver. En este trabajo nos centraremos en el aprendizaje automático inductivo a partir de datos. Para este tipo de aprendizaje, comúnmente se distingue entre aprendizaje no supervisado y aprendizaje supervisado (Bishop, 2013).

En los problemas de aprendizaje no supervisado, se parte de un conjunto de ejemplos para los cuales no conocemos su etiqueta. El objetivo de este tipo de aprendizaje, es identificar grupos de ejemplos (conglomerados o *clústeres*) que sean similares entre sí y diferentes a los ejemplos de otros grupos.

En los problemas de aprendizaje supervisado el objetivo es predecir una característica cuyo valor es desconocido (la etiqueta clase), a partir de otras características conocidas (los atributos que describen a cada uno de los ejemplos). Para ello se parte de una colección de ejemplos caracterizados por sus atributos, y cuyas etiquetas de clase son conocidas. Por tanto, se quiere encontrar patrones que permitan predecir la clase a partir del resto de características. Por ejemplo, imaginemos que tenemos un histórico de personas que han sufrido una enfermedad y sus síntomas. Se quiere encontrar un patrón, que relacione los síntomas con la enfermedad para posteriormente predecir un nuevo caso de dicha enfermedad a partir de sus síntomas.

Existen distintos tipos de problemas a predecir en el aprendizaje automático supervisado, siendo los más habituales los de regresión y clasificación (Sierra, 2006) (Bishop, 2013).

En los problemas de regresión, la etiqueta a predecir toma valores entre los números reales. Por ejemplo, predecir la nota de un examen final de un alumno basándose en el historial de notas de dicho alumno. En los problemas de clasificación, la etiqueta a predecir toma valores discretos, generalmente no ordenados. Las etiquetas clase suelen estar prefijadas. Por ejemplo, si está enferma una persona o no. En este caso, sería una clasificación binaria, ya que solo existen dos posibles valores para la etiqueta. En el caso de que la etiqueta pueda tomar más de dos valores se trata de un problema de clasificación multiclase. Por ejemplo, un problema de diagnóstico médico en el que se intenta determinar la enfermedad a partir de los síntomas.

Para poder predecir la clase de nuevos ejemplos, se utiliza un predictor. El predictor codifica una relación entre los atributos que caracterizan el ejemplo y su clase. La inducción de esta relación a partir de un conjunto de datos etiquetados se realiza mediante un proceso de entrenamiento. Los ciclos de un sistema de reconocimiento de patrones en el aprendizaje automático son los siguientes (Duda, Hart, & Stork, 2012):

1. Recolección de datos para ser utilizados en el proceso de aprendizaje.
2. Selección y construcción de atributos que capturen las características más relevantes de los datos.
3. Elección del algoritmo de aprendizaje. Existen diferentes algoritmos como pueden ser las redes neuronales, vecinos próximos, perceptrón, máquinas de soporte vectorial, etc.
4. Entrenamiento, es la búsqueda de patrones que relacionen las características de los datos con su clase. Como resultado de este proceso se obtiene un predictor.
5. Validación, se valida que el predictor es eficaz para predecir nuevos datos mediante un conjunto de predictores.

1.2 Objetivos

El objetivo de este trabajo es mejorar los sistemas predictivos basados en conjuntos. En concreto, se ampliará la librería EnsembleSVM para generar un conjunto de predictores a partir de ejemplos etiquetados. Como algoritmo de aprendizaje base se utilizan las máquinas de soporte vectorial. Para generar diferentes predictores se aplicarán técnicas de diversificación en el conjunto de datos. Estas técnicas consisten en generar nuevos conjuntos de ejemplos mediante remuestreo (*bagging*) e inyección de ruido en las etiquetas de clase (*class-switching).*

La técnica *bagging*, consiste en generar diferentes predictores a partir de conjuntos de entrenamiento distintos. Cada uno de estos conjuntos es obtenido por remuestreo a partir del conjunto de ejemplos etiquetados inicial. La predicción del conjunto es obtenida por agregación de las predicciones individuales. Se suelen utilizar técnicas de combinación simples, como la media en regresión, o el voto por mayoría en clasificación.

La técnica *class-switching* (Breiman, Randomizing outputs to increase prediction accuracy, 2000) y (Martínez-Muñoz & Suárez, 2005), consiste en generar conjuntos de entrenamiento distintos a partir del original, inyectando ruido en las etiquetas de clase. En concreto, se modifica al azar la etiqueta a un porcentaje de ejemplos seleccionados también de manera aleatoria.

1.3 Estructura del trabajo

La memoria consta de los siguientes capítulos:

- **Capítulo 2.** Estado del arte: En este capítulo se introducen los conceptos básicos del aprendizaje automático supervisado. Se describen asimismo métodos para generar conjuntos de predictores a partir de una colección de ejemplos etiquetados.
- **Capítulo 3**. Proyecto: Se describen los requisitos funcionales y no funcionales del proyecto. Se detallan las diferentes funcionalidades del programa representándolas mediante diagramas de casos de uso. A continuación, se documentan las librerías que se han utilizado para realizar el proyecto mediante un diagrama de clases, en el que se describen las estructuras y las funciones principales utilizadas para generar conjuntos de predictores. Por último, se especifican los módulos por los que está formado el programa.
- **Capítulo 4.** Pruebas y resultados: Se describen los experimentos que se han realizado junto a los resultados obtenidos, acompañados de una interpretación detallada.
- **Capítulo 5.** Conclusiones y trabajo futuro: Se resumen las conclusiones que se derivan del desarrollo del trabajo y de su análisis. Finalmente, se proponen líneas de trabajo futuro en este campo.

2 Estado del arte

En este apartado se introducen los conceptos básicos del aprendizaje automático. Específicamente, se describe cómo generar un predictor mediante un algoritmo de aprendizaje base a partir de datos etiquetados. Por ejemplo, se quiere generar un sistema informático para la tasación automática de viviendas. El motor de este sistema es un predictor, que permite determinar el precio de una vivienda (salida) a partir de sus características (entrada). Para diseñar este predictor, se parte de una base de datos en la que se almacena el registro histórico de las viviendas vendidas. Cada registro en dicha base de datos corresponde a una venta concluida. El registro incluye las características de la vivienda objeto de la transacción y el precio final de venta. A partir de este histórico, el algoritmo de aprendizaje automático permite identificar relaciones que relacionen las características de las viviendas con el precio final de venta. Se trata de un problema de regresión en el que las características de la vivienda corresponden a los atributos. El precio de venta es la etiqueta a predecir.

Existen distintos métodos para determinar la eficacia de un predictor. Idealmente se debería contar con un conjunto etiquetado independiente del utilizado para el entrenamiento de un tamaño suficientemente grande. En la práctica se utilizan distintas técnicas de validación. La versión más básica de este proceso consiste en dividir en dos particiones los ejemplos etiquetados. Una partición se utiliza para entrenar el predictor y la otra partición se utiliza como conjunto de test, para realizar una estimación insesgada del error de predicción. En este trabajo los predictores utilizados son las máquinas de soporte vectorial. En lugar de entrenar un solo predictor, el objetivo es generar un conjunto de predictores. La idea es que, si los predictores son complementarios (es decir, los errores que comenten estos predictores son independientes), la agregación de sus predicciones por métodos de combinación sencillos (por ejemplo, mediante promediado, en problemas de regresión, o mediante voto por mayoría, en problemas de clasificación) debería proporcionar predicciones globales más robustas. Se pueden utilizar diversas técnicas para obtener predictores que sean diferentes. En este trabajo se generarán conjuntos homogéneos, en los que cada predictor se genera aplicando el mismo algoritmo de aprendizaje automático (en este caso, el entrenamiento de la máquina de soporte vectorial) a diferentes conjuntos de entrenamiento generados a partir del conjunto de datos etiquetados inicial mediante técnicas de remuestreo e inyección de ruido en las etiquetas de clase.

2.1 Aprendizaje automático supervisado

El objetivo del aprendizaje supervisado es generar un predictor a partir de un conjunto de ejemplos etiquetados (Bishop, 2013). En su versión más simple, el ejemplo está caracterizado un vector de atributos y una etiqueta de clase. Los valores de los atributos pueden ser de distinto tipo. Por ejemplo, podrían tomar valores reales, enteros, booleanos, texto, etc. El valor de la etiqueta puede ser un valor real (regresión) o uno de entre un conjunto de valores discretos (clasificación). Se llama predictor a una función capaz de etiquetar un elemento a partir de sus características. Al conjunto de ejemplos etiquetados que se utiliza para entrenar el predictor se le denomina datos de entrenamiento:

$D^{train} = \{(\boldsymbol{x_n}, y_n)\}_{n=1}^{N}$ siendo N el número de ejemplos etiquetados

$\boldsymbol{x}_i$ el vector de atributos

y_i la etiqueta, si es de regresión es un valor real,

si es de clasificación $y_i \in (l_1, l_2, \dots, l_C)$

Por ejemplo, se quiere diseñar un sistema para para tasar una vivienda. Para ello, se entrenará un predictor que proporcione como salida el precio estimado de la vivienda a partir de sus características (localidad, zona, metros cuadrados, número de habitaciones, número de cuartos de baño, etc.). Las características forman el vector de atributos y el precio la etiqueta. Los datos de entrenamiento (el registro histórico de viviendas vendidas) se utilizan para entrenar el predictor:

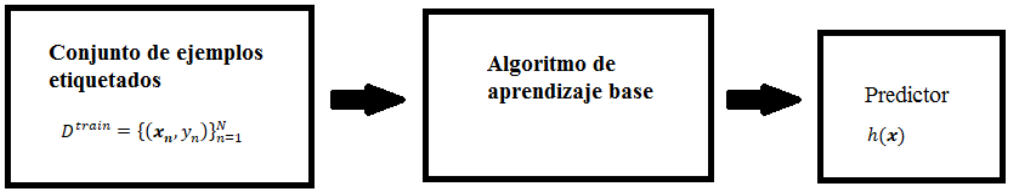

Figura 1. Aprendizaje automático supervisado

Siendo $h(\boldsymbol{x})$ el predictor y $\boldsymbol{x}$ el vector de atributos a predecir la etiqueta

Una manera de abordar el problema de aprendizaje automático es expresar una forma paramétrica para los modelos predictivos. Por ejemplo, en un modelo lineal múltiple, los coeficientes de regresión ($b_0, b_1, b_2, \dots, b_D$) son los parámetros del modelo:

$y = a + b_1x_1 + b_2x_2 +, \dots, +b_Dx_D$ siendo D+1 el número de parámetros.

Los hiperparámetros son valores que se utilizan para configurar el algoritmo de aprendizaje. Por ejemplo, en las máquinas de soporte vectorial, son hiperparámetros el tipo de núcleo que se va a utilizar y su escala, el peso del término de regularización, etc., (ver Anexo A).

Cuando un algoritmo de aprendizaje automático utiliza un modelo erróneo o los valores de los hiperparámetros no están bien elegidos, se puede generar un predictor con sobreajuste o subajuste. Un predictor con sobreajuste o subajuste, tiene una alta probabilidad de predecir de manera errónea una etiqueta de un ejemplo que no se haya utilizado en el conjunto de entrenamiento.

El sobreajuste se da cuando el predictor es complejo y se entrena en exceso, adaptándolo a las características específicas de los ejemplos que se han utilizado para entrenar el predictor. Como se muestra en la siguiente figura, los ejemplos de cada clase

(puntos rojos y azules) han sido separados por un predictor ajustado correctamente (frontera de separación verde) y un predictor con sobreajuste (frontera de separación negra). En la figura 2, se observa como la frontera separadora del predictor sobreajustado es muy compleja.

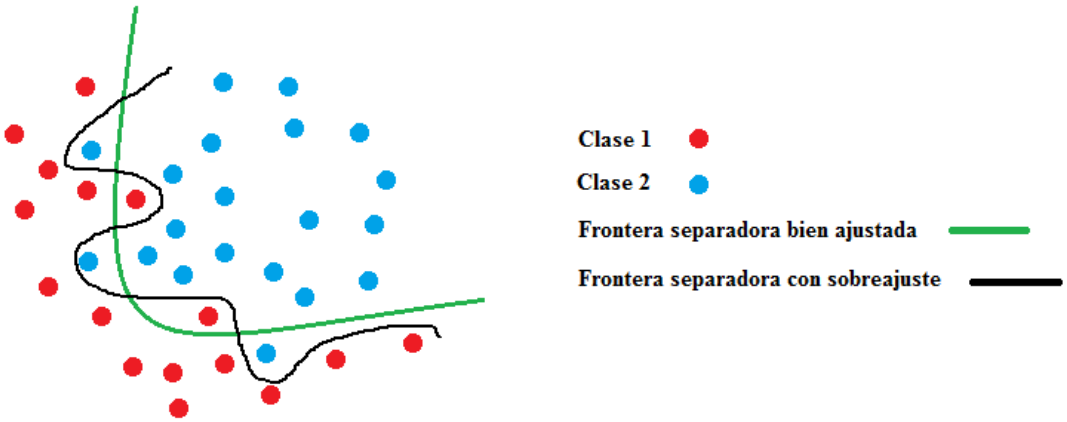

Figura 2. Predictor con sobreajuste

El subajuste (underfitting) es lo contrario que el sobreajuste. Se produce cuando tenemos un problema muy complejo y el algoritmo de aprendizaje automático genera un predictor excesivamente simple. Como se muestra en la siguiente figura, los ejemplos de cada clase están separados por un predictor ajustado correctamente (la línea verde) y un predictor con subajuste (la línea negra). Se observa como la frontera separadora del predictor con subajuste es muy simple.

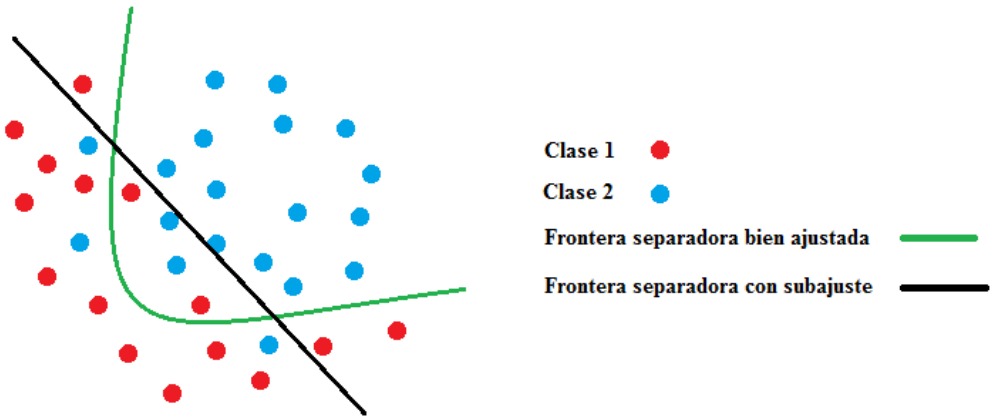

Figura 3. Predictor con subajuste

Como se verá en el siguiente apartado, el sobreajuste y el subajuste se pueden intentar evitar ajustando y validando un predictor mediante validación.

Un conjunto de test está compuesto por ejemplos independientes de los de entrenamiento.

Consideremos el conjunto $D^{test} = \{(\boldsymbol{x}_n^{test}, y_n^{test})\}_{n=1}^{N^{test}}$ de N^{test} datos de test.

El proceso de clasificación consiste en calcular $h(\boldsymbol{x}^{test})$, la salida del predictor basada en la descripción del ejemplo mediante el vector de atributos $\boldsymbol{x}^{test}$. La predicción es correcta si la etiqueta predicha coincide con la real del ejemplo y errónea en caso contrario:

$$(\boldsymbol{x}^{test}, y^{test}) \rightarrow h(\boldsymbol{x}^{test})$$

Si $h(\boldsymbol{x}^{test}) = y^{test}$ acierto

Si $h(\boldsymbol{x}^{test}) \neq y^{test}$ error

La tasa de error en el conjunto de test (error de test) es la fracción de ejemplos del conjunto de test en los que la predicción es errónea:

$$E^{test} = \frac{1}{N^{test}} \sum_{n=1}^{N^{test}} h(\boldsymbol{x}_n^{test}) \neq y_n^{test}$$

2.1.1 Técnicas de validación en aprendizaje automático

La validación es una técnica de análisis estadístico, utilizada para estimar la tasa de error, que es el porcentaje de predicciones erróneas. En el proceso de validación se dividen los ejemplos etiquetados de forma que un subconjunto de estos ejemplos es usado para entrenar y otro (normalmente el complementario) para validar (Schneider, 1997).

Las técnicas de validación pueden ser también utilizadas para determinar el modelo de predicción (por ejemplo, redes neuronales o SVMs), su arquitectura (por ejemplo, capas ocultas y neuronas en cada capa oculta en redes neuronales, o tipo de kernel en SVMs) y para seleccionar los valores de los hiperparámetros de un algoritmo de aprendizaje automático. En concreto, para determinar los hiperparámetros, se generan predictores configurados con diferentes valores de hiperparámetros: Se define una secuencia de valores para cada hiperparámetro (desde un valor mínimo hasta un valor máximo del hiperparámetro en una escala que podría ser logarítmica), y generar un predictor por cada una de las combinaciones de los valores de los hiperparámetros. Finalmente se seleccionan los valores de los hiperparámetros que minimizan el error estimado por validación.

A continuación, se describe los distintos tipos de validación utilizados de manera habitual:

- **Validación aleatoria:** Consiste en dividir el conjunto de ejemplos etiquetados en dos subconjuntos, uno de entrenamiento y otro de validación. Cada uno de los ejemplos es asignado de manera aleatoria a uno de estos conjuntos. Esta técnica de validación, es la menos costosa computacionalmente. El principal inconveniente es que los predictores son entrenados solo con un subconjunto de los datos. Dado que las particiones son aleatorias, las estimaciones del error pueden tener mucha variabilidad. Para mitigar estas fluctuaciones pueden realizar diferentes particiones y promediar las tasas de error resultantes:

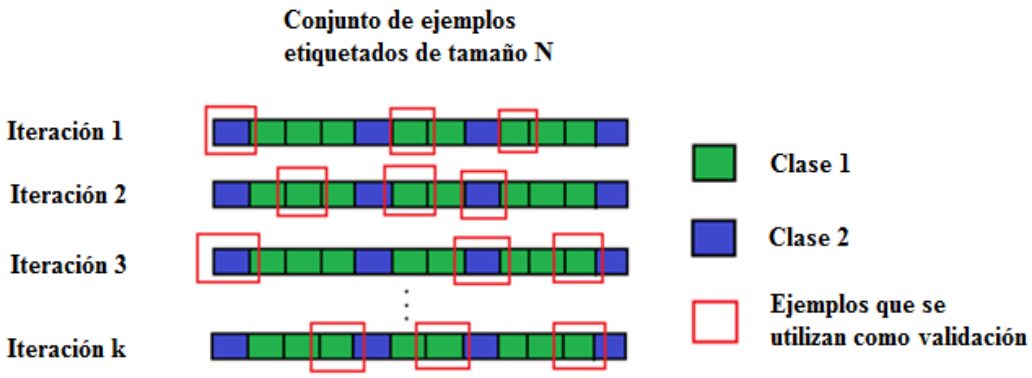

Figura 4. Validación cruzada aleatoria

- **Validación cruzada de k particiones:** El conjunto de ejemplos etiquetados, se divide en k subconjuntos distintos. En cada iteración, uno de los subconjuntos, es utilizado para estimar el error cometido por un predictor entrenado con los ejemplos etiquetados del resto de subconjuntos. La estimación por validación cruzada del error de clasificación, es el promedio de estas k estimaciones.
Por ejemplo, para k = 3:

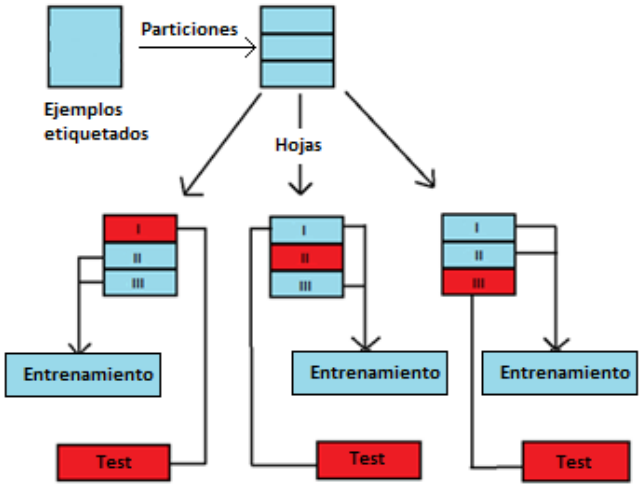

Figura 5. Validación cruzada de k = 3 hojas

Como se muestra en la figura 5, en la primera hoja se utiliza la partición II y III para realizar el entrenamiento del algoritmo de aprendizaje y la partición I para realizar la validación dando como resultado:

E_I : Error en subconjunto I del predictor entrenado con los subconjuntos II y III

Se aplica para cada hoja:

E_{II} : Error en subconjunto II del predictor entrenado con los subconjuntos I y III

E_{III} : Error en subconjunto III del predictor entrenado con los subconjuntos I y II

El error final se calcula, como la media aritmética de los errores obtenidos en cada hoja:

$$E = \frac{1}{k}\sum_{i=1}^{k} E_i$$

- **Validación cruzada dejando uno fuera:** Este caso corresponde a una validación cruzada en la que existen tantas hojas (k) como numero de ejemplos en el conjunto (N). En cada iteración, se utiliza un ejemplo como validación y el resto de ejemplos para el entrenamiento. La estimación por validación cruzada del error de clasificación es el promedio de estas N estimaciones:

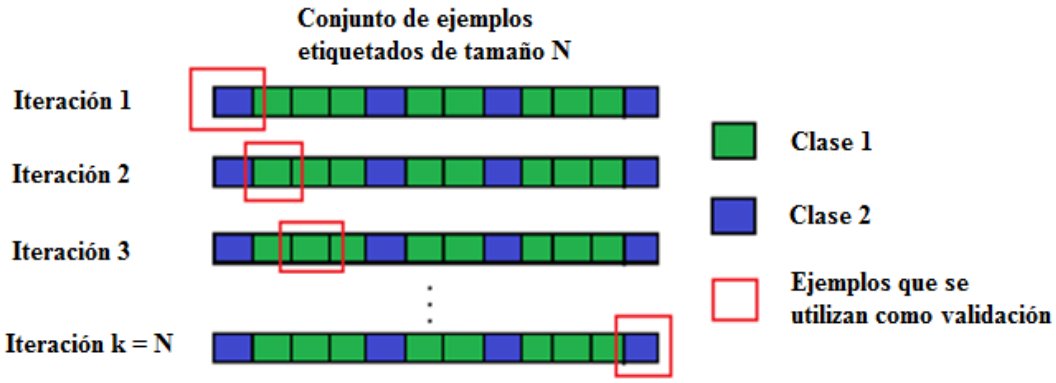

Figura 6. Validación cruzada dejando uno fuera k = N

El error total viene dado por:

$E = \frac{1}{N}\sum_{n=1}^{N} E_n$ siendo $E_n$el error del predictor en cada hoja

La validación cruzada dejando uno fuera es muy costosa ya que tiene que realizar N iteraciones, en cada una de las cuales es necesario entrenar un predictor distinto. Por este motivo, se utilizará en este trabajo la validación cruzada de k particiones.

2.2 Máquinas de soporte vectorial

Las máquinas de soporte vectorial (En inglés "*Support Vector Machine*", SVM) son un conjunto de algoritmos de aprendizaje automático supervisado cuyo objetivo es encontrar un hiperplano para la separación de clases en un espacio extendido por maximización del margen de clasificación. La frontera de separación entre las clases es lineal en el espacio extendido, pero no lineal en el espacio original. El algoritmo fue desarrollado por Vladimir Vapnik y su equipo en los años 90 (Cortes & Vapnik, 1995). Tuvo una gran repercusión debido a su sólido fundamento teórico (están basadas en el principio de minimización del riesgo estructural introducido por Vapnik) y por los excelentes resultados que se obtienen en numerosos problemas de interés práctico. Estas máquinas pueden ser aplicadas en problemas de clasificación y de regresión. En la

actualidad se utilizan con éxito en campos como el reconocimiento de imágenes y textos, la recuperación de información dañada, en aplicaciones biomédicas o en la mejora de motores de búsqueda en internet (Joachims, Search Engines that Learn from Implicit Feedback, 2007).

En el entrenamiento de una SVM el objetivo es reducir el riesgo estructural. Para ello se considera una familia de modelos de complejidad creciente y se selecciona el de mínima complejidad capaz de describir los datos de entrenamiento. En problemas de clasificación binaria, se busca un predictor lineal en un espacio extendido que maximice el margen de separación entre las clases.

En los siguientes apartados, analizaremos por separado el caso de problemas de clasificación binaria separables linealmente y de problemas no separables linealmente (Bishop, 2013).

2.2.1 SVM para problemas separables linealmente

Consideremos el problema de inducir una SVM a partir de un conjunto de datos etiquetados:

$$D^{train} = \{(\mathbf{x}_i, t_i)\}_{i=1}^{n} \text{ con } t_i \in \{-1,1\}$$

Suponiendo que el problema es separable linealmente y que no tiene ruido, el objetivo es construir un hiperplano que se encuentra a la mínima distancia de los ejemplos más cercanos de cada clase (Joachims, Learning to Classify Text Using Support Vector Machines, 2013).

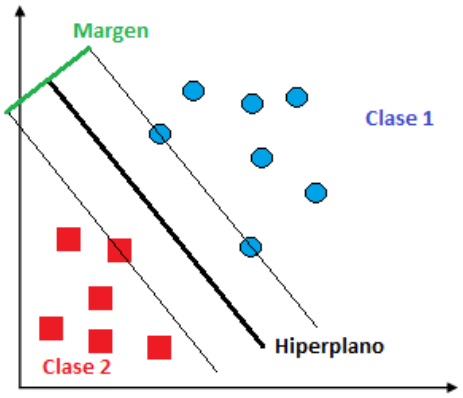

Figura 7. Hiperplano de separación óptimo y margen máximo

Un hiperplano h de dimensión (d-1) en un espacio de d dimensiones está descrito por la ecuación:

$$h(\mathbf{x}) = 0, \text{ donde } h(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + b \text{ , } \mathbf{w} \text{ y } \mathbf{x} \in R^d \text{ y b es el término de sesgo.}$$

Dado que el conjunto de ejemplos etiquetados es de tamaño finito, existen infinitos hiperplanos que separan las clases. Para escoger el hiperplano óptimo, en el sentido de

que la regla que lo utilice como frontera de decisión sea la que mejor generalice, dentro de los clasificadores lineales, se escoge aquél que minimiza el margen. En problemas sin solapamiento de clases, el margen viene dado por la distancia entre el hiperplano de separación y los ejemplos de ambas clases más cercanos a él. Estos ejemplos son los denominados vectores soporte.

La distancia de un punto $\boldsymbol{x}$ al hiperplano h es:

$$dist(h, \boldsymbol{x}) = \frac{|\boldsymbol{w}^t \boldsymbol{x} + b|}{||\boldsymbol{w}||}$$

donde $||\boldsymbol{w}||$ es la norma euclídea $||\boldsymbol{w}|| = \sqrt{w_1^2 + w_2^2 + \cdots + w_D^2}$

Observando que cualquier escalado $\boldsymbol{w} \rightarrow c\boldsymbol{w}$ y $b \rightarrow c\, b$ con c > 0 mantiene la distancia $dist(h, \boldsymbol{x})$ constante, podemos elegir que la distancia de los puntos más cercanos al hiperplano (los vectores soporte) sea:

$$dist(h, \boldsymbol{x}_i^*) = \frac{1}{||\boldsymbol{w}||}$$

Con $|\boldsymbol{w}^t \boldsymbol{x}_i^* + b| = 1$ para estos puntos. Para el resto de puntos de la muestra

$$dist(h, \boldsymbol{x}_i) \geq 1$$

El problema a resolver es una optimización convexa y posee una única solución. Utilizando la formulación del problema con:

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i \, t_i \boldsymbol{x_i}$$

Este se puede resolver maximizando la función Lagrangiana:

$$\tilde{L}(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j t_i t_j \boldsymbol{x}_i^t \boldsymbol{x_j} \quad \text{con } \alpha_i \geq 0 \text{ y } \sum_{i=1}^{n} \alpha_i \, t_i = 0$$

Aplicando las condiciones Karush-Kuhn-Tucker se obtiene:

$$\alpha_i \geq 0$$
$$t_i(\boldsymbol{w}^t x_i + b) - 1 \geq 0$$
$$\alpha_i[t_i(\boldsymbol{w}^t x_i + b) - 1] \geq 0$$

De acuerdo con estas condiciones, existen dos tipos de puntos

- Los puntos para los cuales $\alpha_i = 0$ son los vectores soporte que definen el margen:

$$t_i(\boldsymbol{w}^t \boldsymbol{x_i} + b) = 1$$

- Los puntos para los cuales $\alpha_i > 0$ están fuera del margen:

$$t_i(\boldsymbol{w}^t \boldsymbol{x_i} + b) > 1$$

Finalmente, el hiperplano de separación que maximiza el margen es:

$$\mathrm{h}(\mathbf{x}) = \boldsymbol{w}^t \boldsymbol{x}_i + b = \sum_{i=1}^{n} t_i \alpha_i \boldsymbol{x}_i{}^t \mathbf{x} + b$$

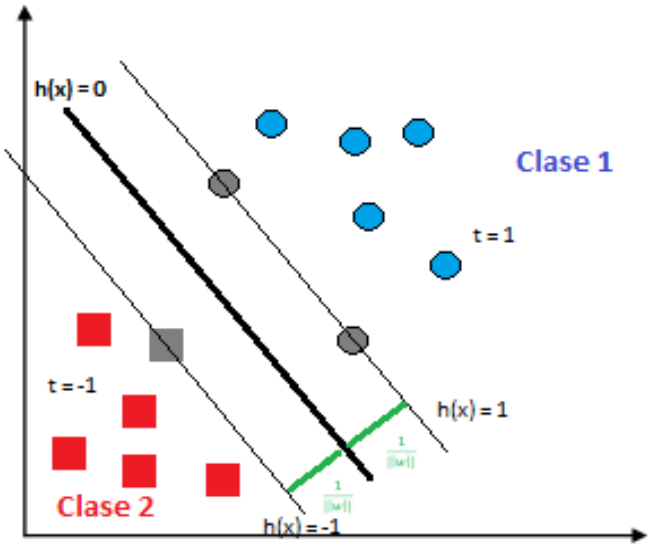

Figura 8. Distancia del hiperplano a cualquier punto y vectores soporte

2.2.2 SVM para problemas no separables linealmente

En el caso de problemas no separables linealmente, se puede formular el problema en un espacio de características $\phi(\mathbf{x})$ de dimensión elevada (posiblemente infinita) en el que sea posible separar las clases mediante un hiperplano. Esta formulación es posible gracias al llamado "truco del núcleo" ("*kernel trick*").

La función núcleo representa el producto escalar en el espacio de características $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^t \phi(\mathbf{x}_j)$

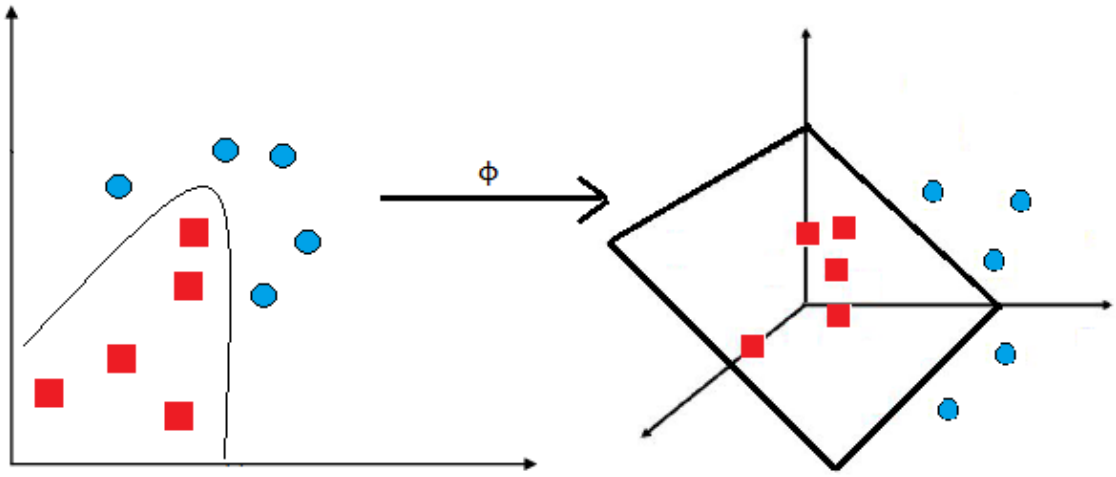

Figura 9. Función de decisión no lineal (hiperplano) a nuevas dimensiones en el espacio aplicando la función núcleo

En su versión lineal el hiperplano de margen máximo se obtenían como se ha mostrado anteriormente con:

$$\mathrm{h}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i t_i \boldsymbol{x}_i{}^t \mathbf{x} + b$$

Para obtener la frontera de decisión en una SVM no lineal, se sustituye el producto escalar en el espacio de entrada ($\boldsymbol{x}_i^t \mathbf{x}$) por el núcleo:

$$\mathrm{h}(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i t_i \mathrm{k}(\boldsymbol{x}_i, \mathbf{x}) + b$$

A veces los conjuntos de datos pueden estar contaminadas con ruido. Esto se puede incorporar en el modelo considerando una variable de holgura ξ, que permite errores en la vecindad del hiperplano de separación:

$$t_i(\boldsymbol{w}^t \boldsymbol{x}_i + b) \geq 1 - \xi_\mathrm{i}$$
$$\xi_\mathrm{i} \geq 0$$

Si $\xi_\mathrm{i} > 1$ son puntos que están mal clasificados, si $0 \leq \xi_\mathrm{i} \leq 1$ serán puntos bien clasificados dentro del margen y si $\xi_\mathrm{i} = 0$ estarán bien clasificados fuera del margen.

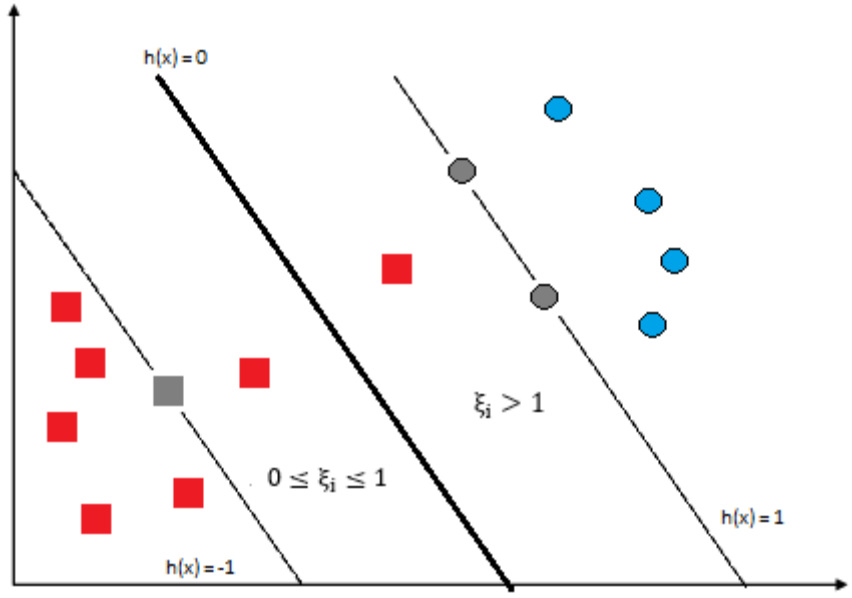

Figura 10. Clasificación de ejemplos en una SVM no lineal

Se pueden aplicar varios tipos de núcleos (ver Anexo A). En este trabajo utilizaremos sobre todo SVMs con núcleo Gaussiano

$$K(\boldsymbol{x}, \boldsymbol{x}') = exp(-\gamma||\boldsymbol{x} - \boldsymbol{x}'||^2)$$

El parámetro γ (Gamma) es la anchura del núcleo. Determinar su valor con precisión es muy importante para diseñar SVMs con buena capacidad de generalización. En general si γ es demasiado grande la SVM tiene tendencia al sobreajuste. Si el valor de γ es demasiado pequeño la frontera de decisión es excesivamente rígida, y el modelo será muy pobre.

Otro parámetro muy importante en el entrenamiento de una SVM es el peso del término de regularización (C). Este término permite que se puedan cometer errores a la hora de clasificar ejemplos con la SVM. Cuanto mayor es el valor C, la penalización de los errores es más severa. Si C disminuye, se permite un mayor porcentaje de errores. En el primer caso se tiende a un sobreajuste. En el segundo caso al subajuste.

El valor de estos parámetros se suele determinar por validación cruzada. El rango de valores que se explora para γ está entre $(2^{-15}, \ldots, 2^{3})$. El valor óptimo del parámetro de regularización (C) se suele buscar entre $(2^{-5}, \ldots, 2^{15})$.

2.3 Conjunto de predictores

En aprendizaje automático los conjuntos son sistemas que combinan las salidas de diferentes predictores para realizar una predicción mejorada (Dietterich, 2000).

La técnica *bagging* (*Bootstrap sampling and aggregation*), consiste en generar conjuntos de ejemplos diferentes, mediante remuestreo de los ejemplos del conjunto de entrenamiento inicial. Dado un conjunto de ejemplos etiquetados $D_{train} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$, se generan tantos conjuntos por remuestreo como predictores se desee generar. El remuestreo se puede hacer con o sin reemplazo. Con reemplazo, los ejemplos que se

utilizan se vuelven a insertar en el conjunto. Sin reemplazo, los ejemplos que se utilizan, no se vuelven a insertar en el conjunto. Si se utiliza remplazo el tamaño del conjunto que se genera, D_t, suele ser el mismo que el conjunto original, D_{train}. En el conjunto D_t, generado por remuestreo, cada ejemplo de D_{train} tiene una probabilidad de aparecer de $1-\left(1-\frac{1}{N}\right)^N$, valor que se aproxima al $1-\frac{1}{e}=63\%$ de los ejemplos cuando N tiende a infinito. Si no se utiliza reemplazo para generar los conjuntos, se suelen generar con un tamaño del 50% del tamaño de D_{train}. Este procedimiento se repite T veces, hasta obtener los T predictores. Cada conjunto de ejemplos creado, tendrá el mismo número de ejemplos (M), pudiéndose repetir cada ejemplo en el mismo conjunto. Dependiendo si el procedimiento se hace con reemplazo o sin reemplazo (Efron & Tibshirani, 2000).

Para t = 1 hasta T

Se extrae con o sin reemplazo de los ejemplos $\{1,2,3 \dots N\} \rightarrow \{n_1^t, n_2^t, \dots n_M^t\}$

Se construyen los nuevos conjuntos:

$$\mathrm{D_{train}} = \{(\boldsymbol{x_n}, y_n)\}_{n=1}^{N} \rightarrow D_{train}^{t} = \left\{\left(\boldsymbol{x_{n_m^t}}, y_{n_m^t}\right)\right\}_{m=1}^{M}$$

La predicción del conjunto es obtenida por agregación de las predicciones individuales. Se suelen utilizar técnicas de combinación simples, en este caso se va a utilizar el voto por mayoría ya que se van a resolver problemas de clasificación.

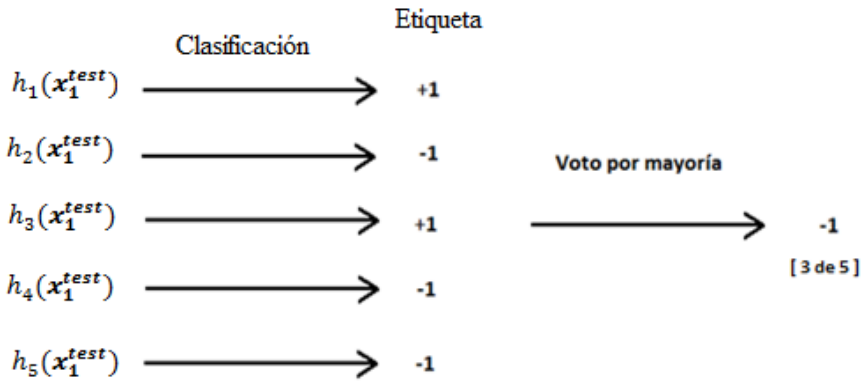

Figura 11. Etiquetado de un ejemplo por un conjunto de predictores con *T = 5*

Para todos los problemas, el error de un conjunto de predictores viene dado por:

$$D^{test} = \{(\mathbf{x_n^{test}}, y_n^{test})\}_{n=1}^{\mathrm{N_{test}}}$$

$$E^{test} = \frac{1}{N_{test}} \sum_{n=1}^{N_{test}} \mathbb{I}(h(\boldsymbol{x}_n^{test}) \neq y_n^{test})$$

Donde $\mathbb{I}(True) = 1$ e $\mathbb{I}(Falso) = 0$.

Otro método de generación de conjuntos es *boosting*. En *boosting*, se entrenan iterativamente una serie de predictores. Cada predictor nuevo que se entrena se centra en los ejemplos mal clasificados por el predictor anterior (Freund & Schapire, 1997).

En este trabajo, se quiere mejorar los sistemas de predicción basados en conjuntos de predictores, concretamente de máquinas de soporte vectorial. Para realizar esta mejora, se va a utilizar otra técnica de remuestreo de datos además de *bagging*, llamada *class-switching*.

La técnica de remuestreo *class-switching*, se basa en inyectar ruido en las etiquetas clase de una fracción de los ejemplos del conjunto, seleccionados de manera aleatoria. Por ejemplo, en un problema con dos clases $\{l_1, l_2\}$ si $\mathbf{x}_\mathrm{m}$ está etiquetado con la clase $y_m = l_1$ y pertenece al porcentaje de ejemplos a cambiar la clase, se etiqueta con $y_m = l_2$. Si $\mathbf{x}_\mathrm{m}$ está etiquetado con $y_m = l_2$ y pertenece al conjunto de ejemplos a los que se modifica la clase, se etiqueta con $y_m = l_1$. Si no pertenece al porcentaje de ejemplos a cambiar la clase, mantendrá su etiqueta.

3 Proyecto

En este capítulo se analizan los requisitos funcionales, no funcionales, así como todas y cada una de las decisiones que se han tomado para desarrollar el proyecto.

Cada requisito funcional se representa en un diagrama de flujo. Se da cumplida explicación de todos los formatos de ficheros de entrada, parámetros de entrada necesarios para un uso correcto del programa y ficheros de salida que genera el mismo. Por último, se muestran todas las funcionalidades que puede realizar un usuario en el programa mediante casos de uso.

Detallaré las librerías utilizadas en el programa con una explicación de las principales clases y funciones que se utilizan para generar un conjunto de predictores y etiquetar nuevos ejemplos. Para concluir, detallaré cómo se ha implementado la ampliación de la librería para generar un conjunto de predictores utilizando la técnica de remuestreo *class-switching*.

3.1 Definición y análisis de requisitos

En este apartado, se analizan los requisitos funcionales y no funcionales. El análisis de requisitos se define como "el proceso de estudio de las necesidades de los usuarios para llegar a una definición de los requisitos del sistema, de hardware o de software, así como el proceso de estudio y refinamiento de dichos requisitos" (IEEE Standard Glossary of Software Engineering Terminology, 1990).

- **Requisito funcional:** Definición de los servicios que el sistema debe proporcionar, reacción esperada a entradas particulares y comportamiento en situaciones particulares. Describen lo que el sistema debe hacer. Dependen del tipo de software que se desarrolle y de la utilización que quieran darle los usuarios (Sommerville, 2005).

- **Requisito no funcional:** Restricciones que afectan a los servicios o funciones del sistema, tales como restricciones de tiempo, rendimiento, usabilidad, mantenibilidad, coste, interfaz… (Sommerville, 2005).

A partir de cada requisito funcional se representa mediante un diagrama de flujo dicho requisito indicando los flujos de datos de entrada y las salidas que devuelve el algoritmo de aprendizaje automático.

3.1.1 Requisitos funcionales

- **RF-01:** Se insertan como entradas del programa un conjunto de ejemplos etiquetados D_{train} y una configuración de las máquinas de soporte vectorial (como puede ser hiperparámetros, porcentaje de remuestreo, porcentaje de *class-switching*, número de predictores). Como salida, el programa crea un conjunto de predictores.

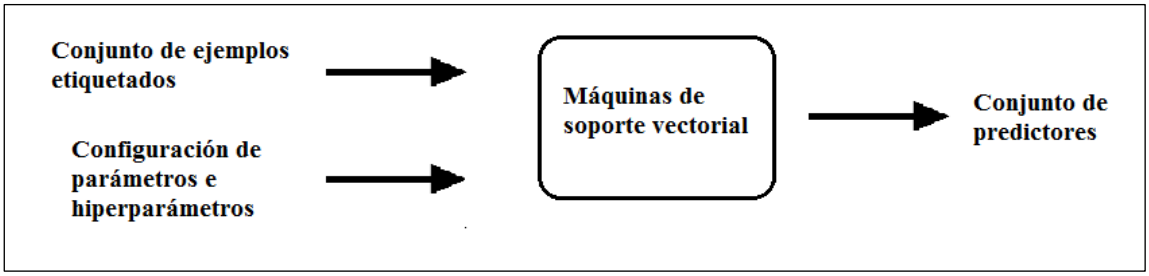

Figura 12. DFD-01, Crear un conjunto de predictores

- **RF-02:** Se introducen como entradas del programa un conjunto de predictores y un conjunto de ejemplos etiquetados D_{test}. Como salida, el programa devuelve la tasa de error E^{test}del conjunto D_{test} .

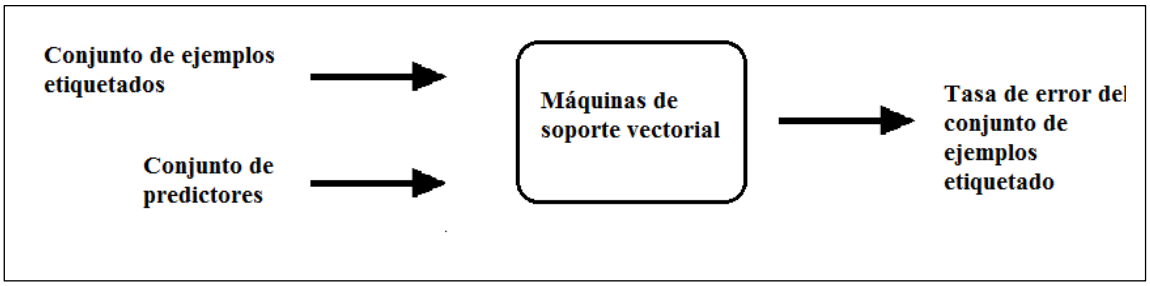

Figura 13. DFD-02, Tasa de error del conjunto de ejemplos etiquetados

- **RF-03:** Se insertan como entradas del programa un conjunto de predictores y un conjunto de ejemplos sin etiquetar D_{test}. Como salida, el programa devuelve un fichero con las etiquetas del conjunto de datos D_{test}.

Figura 14 . DFD-03, Etiquetado del conjunto de ejemplos sin etiquetar

- **RF-04:** Se introducen como entradas del programa un conjunto de ejemplos etiquetados que el programa utilizará como datos de entrenamiento D_{train}, y otro conjunto de ejemplos etiquetados que el programa utilizará como datos de test D_{test}. Devuelve como salida, la tasa de error E^{train} de los datos de entrenamiento D_{train}, y la tasa de error E^{test}de los datos de test D_{test}.

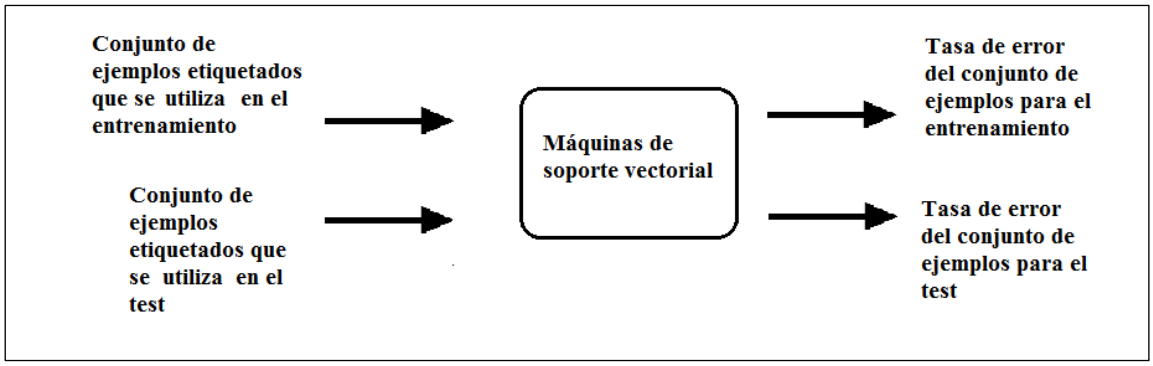

Figura 15. DFD-04, Tasas de error del conjunto de ejemplos de entrenamiento y de test

- **RF-05:** Como entradas del programa se introducen un conjunto de ejemplos etiquetados D_{train} y un conjunto de ejemplos sin etiquetar D_{test}. El programa devuelve como salida, el conjunto de ejemplos D_{test}etiquetado.

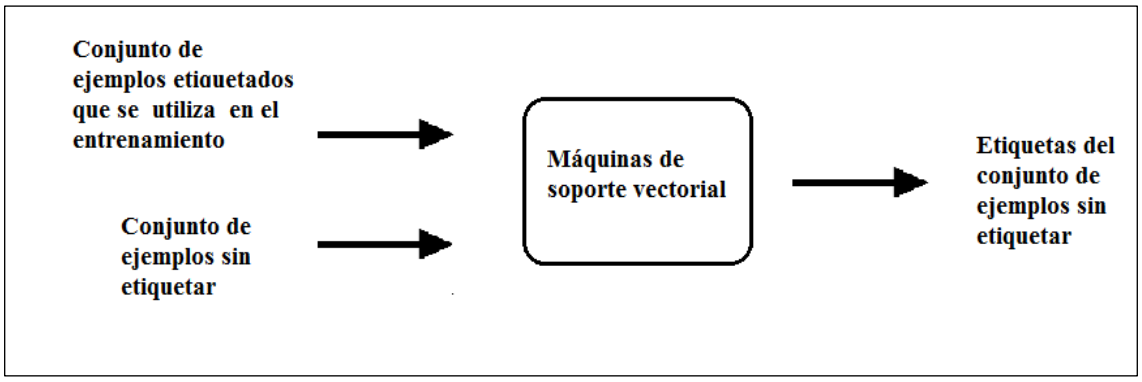

Figura 16 . DFD-05, Etiquetado del conjunto de ejemplos sin etiquetar

- **RF-06:** Se insertan como entradas del programa un conjunto de ejemplos etiquetados D_{train}, otro conjunto de ejemplos etiquetados D_{test}, y una configuración del algoritmo de aprendizaje. Como salida, el programa devuelve la tasa de error E^{test}de los ejemplos de test D_{test}, y el error E^{train}de los ejemplos etiquetados D_{train}.

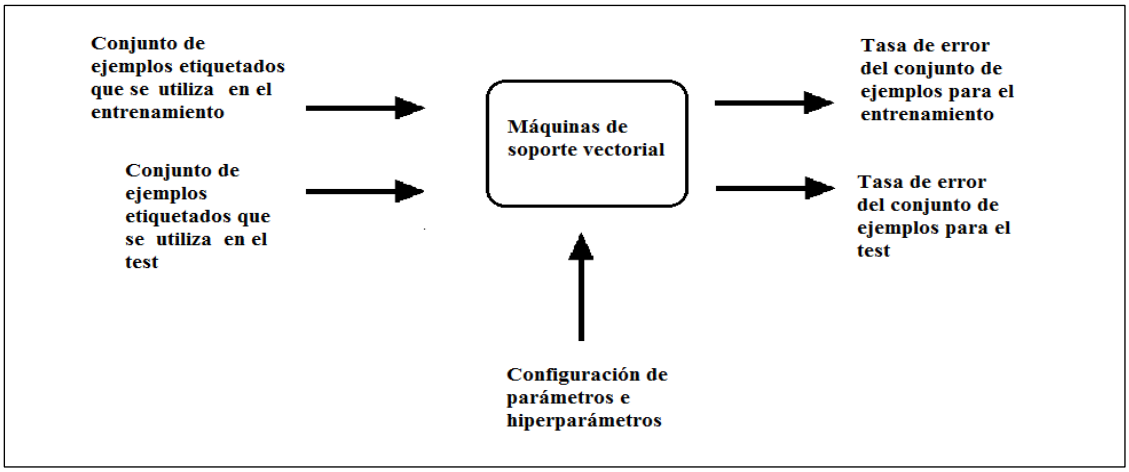

Figura 17. DFD-06, Tasa de error del conjunto de ejemplos etiquetados

- **RF-07:** Se insertan como entradas del programa un conjunto de ejemplos etiquetados D_{train}, otro conjunto de ejemplos sin etiquetar D_{test}, y una configuración del algoritmo de aprendizaje. Como salida del programa, se obtiene el conjunto datos de test D_{test} etiquetado.

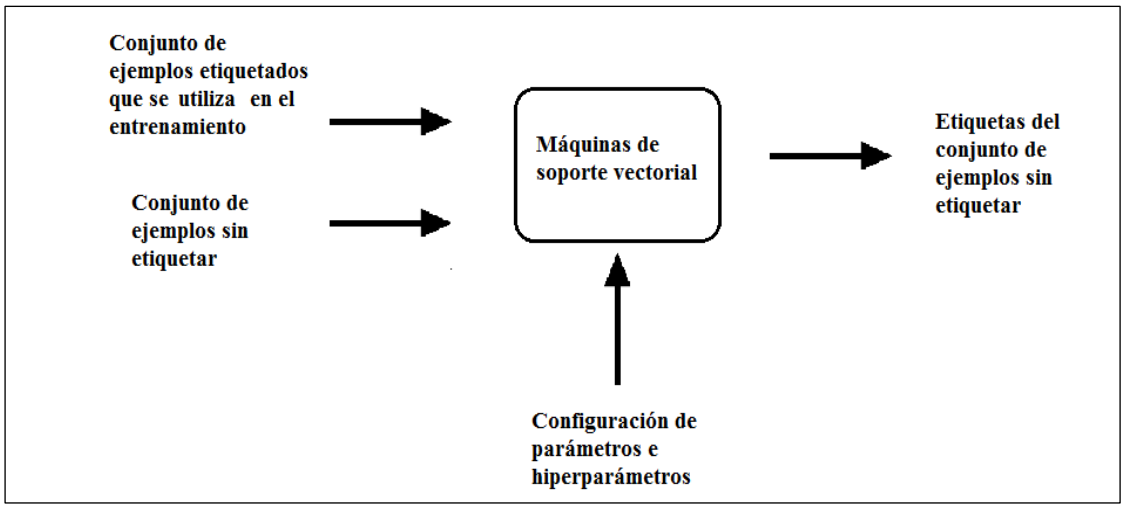

Figura 18. DFD-07, Etiquetado del conjunto de ejemplos sin etiquetar

3.1.2 Requisitos no funcionales

- **RNF-01:** El programa se podrá ejecutar desde una terminal de manera sencilla. Debe indicar al usuario cómo proceder paso a paso para obtener un conjunto de predictores y realizar la clasificación de un conjunto ejemplos.

- **RNF-02:** Se va a utilizar el mismo estilo de programación que en la librería EnsembleSVM, para poder facilitar su mantenibilidad y ampliación de la librería. El leguaje que se utiliza es C++.

3.2 Análisis

En este apartado se explicarán todos los casos de uso que un usuario puede realizar en el programa. Se realizará una introducción a las librerías que se han utilizado para el desarrollo del programa. Para entender mejor como se ha estructurado el programa, se mostrarán y explicarán las clases, y un detalle las principales funciones que se han utilizado para desarrollar la ampliación de la librería.

3.2.1 Casos de uso

En este apartado, se van a representar y explicar diferentes casos de uso. Se describe, el modo en que un usuario interactúa con el programa, así como, todas las actividades que puede realizar el usuario, y en caso de error, como procede el programa.

Actor: Usuario
Rol: Crear un conjunto de predictores

- El usuario ejecuta el programa indicando como parámetros de entrada el nombre del fichero que contiene el conjunto de datos etiquetados, el porcentaje de remuestreo, el porcentaje de clases a cambiar y el valor de los hiperparámetros C y γ.
- El programa, verifica que los hiperparámetros y parámetros de entrada son correctos.
- En caso de error el programa muestra una alerta visual por pantalla. Muestra el nombre del hiperparámetro o parámetro en el que se ha producido el error.
- En caso de ser todos los hiperparámetros y parámetros de entrada correctos, el programa devolverá como salida un fichero con un conjunto de predictores.

Actor: Usuario
Rol: Tasa de error del conjunto de ejemplos etiquetados

- El usuario ejecuta el programa indicando como parámetros de entrada el nombre del fichero que contiene el conjunto de datos etiquetados o el nombre del fichero con el conjunto de predictores.
- El programa verifica que los parámetros de entrada son correctos.
- En caso de error el programa muestra una alerta que detalla en que parámetro se ha producido el error.
- En caso de estar todos los parámetros de entrada bien, el programa devolverá como salida, la tasa de error de la predicción de la etiqueta clase, por el conjunto de predictores.

Actor: Usuario
Rol: Etiquetado del conjunto de ejemplos sin etiquetar

- El usuario ejecuta el programa indicando como parámetros de entrada el nombre del fichero con el conjunto de datos sin etiquetar, el nombre del fichero con el conjunto de predictores o el nombre del fichero con el conjunto de entrenamiento.
- El programa verifica que los parámetros de entrada son correctos.
- En caso de error muestra una alerta que detalla el nombre del parámetro erróneo.

- En caso de que los parámetros sean correctos, el programa devolverá como salida un fichero con las etiquetas del conjunto de ejemplos sin etiquetar.

Actor: Usuario
Rol: Tasas de error del conjunto de ejemplos de entrenamiento y de test

- El usuario ejecuta el programa indicando como parámetros de entrada el nombre del fichero que contiene el conjunto de ejemplos etiquetados de entrenamiento, y el nombre del fichero con el conjunto de ejemplos etiquetados de test.
- El programa verifica que los parámetros de entrada son correctos.
- En caso de error, el programa muestra una alerta. En la alerta se detalla el nombre del parámetro erróneo.
- En caso de que todos los parámetros sean correctos, el programa devolverá las tasas de error de la predicción de la etiqueta clase, de los ejemplos del conjunto de entrenamiento y de test.

3.3 Diseño e implementación

En este apartado lo primero que se menciona son las librerías LIBSVM (Chang & Lin, 2011) y EnsembleSVM (Claesen, 2014). Estas librerías se han tomado como referencia y utilizado en este trabajo. Son librerías de uso abierto que recogen la funcionalidad principal para generar conjuntos de predictores y etiquetar conjuntos de datos. Para dicha generación, las librerías utilizan como algoritmo de aprendizaje base las máquinas de soporte vectorial.

Una vez que se han mencionado las librerías, se definen los diferentes módulos que tendrá el programa. Se detallan las principales clases y funciones utilizadas para generar conjuntos de predictores utilizando las máquinas de soporte vectorial mediante remuestreo con *bagging* y *class-switching*.

3.3.1 LIBSVM

La librería LIBSVM (Chang & Lin, 2011) es un software de código abierto. Fue desarrollada en la Universidad Nacional de Taiwán por Chih-Chung Chang y Chih-Jen Lin. Como lenguaje de programación para escribir la librería utilizaron C++ y Java. Hace uso de la licencia BSD (*Berkeley Software Distribution*) (ver más en detalle Anexo B).

Esta librería se utiliza para resolver problemas predictivos mediante predictores que utilizan como algoritmo base las máquinas de soporte vectorial. En la librería se implementan diferentes tipos de máquinas de soporte vectorial, de esta manera se pueden resolver problemas de clasificación y de regresión. En LIBSVM se pueden seleccionar diferentes tipos de núcleos (ver Anexo A) para entrenar un predictor. También la librería nos permite utilizar la validación cruzada, para validar el predictor, y obtener los valores de los hiperparámetros que utilizan las máquinas de soporte vectorial.

La librería LIBSVM ofrece interfaces para una gran variedad de lenguajes de programación (Python, R, MATLAB, Perl, Ruby, Weka, Common LISP, CLISP, Haskell, OCaml, LabVIEW y PHP).

Para poder trabajar con la librería y que tenga un funcionamiento adecuado es necesario que los ficheros de texto que se utilicen estén en el formato Sparse CSV (ver Anexo D). Estos datos de texto, son los ejemplos etiquetados o sin etiquetar que se van a utilizar en el programa, para entrenar un predictor y que este etiquete nuevos ejemplos.

Por último, la librería nos permite hacer uso de funciones para predecir la etiqueta clase de ejemplos que no se han utilizado para entrenar el predictor. Como salida, el programa genera un fichero con las etiquetas predichas por el predictor.

3.3.2 EnsembleSVM

La librería EnsembleSVM (Claesen, 2014) es un proyecto de aprendizaje automático desarrollado en el lenguaje de programación C++. Hace uso de la licencia pública general GNU LGPL (ver más en detalle en el Anexo B).

EnsembleSVM se utiliza para crear un conjunto de predictores mediante máquinas de soporte vectorial. La librería EnsembleSVM, engloba la librería LIBSVM para utilizar las principales funciones que generan un predictor. El objetivo de la librería EnsembleSVM es resolver problemas de rendimiento, como pueden ser de memoria y de tiempo, en problemas de aprendizaje automático de gran coste computacional.

Para generar los conjuntos de ejemplos que usan los predictores para entrenar, la librería EnsembleSVM nos ofrece la técnica de remuestreo *bagging*. Para que el programa funcione adecuadamente los ficheros de entrada tienen que estar en un formato permitido (se pueden ver en el Anexo D).

El enfoque principal de la librería está en el tiempo en el que el programa entrena el conjunto de predictores. EnsembleSVM reduce el tiempo de entrenamiento, ya que se ha implementado con multihilo. La utilización del multihilo permite que los predictores se entrenen en paralelo.

EnsembleSVM está implementada en C++, requiere ser compilada con un compilador reciente como gcc ≥ 4,7 o clang ≥3,2. Para la generación de los ejecutables de las herramientas que ofrece la librería, se ha implementado un Makefile basado en GNU (Gough, 2005).

En esta librería se va a realizar la ampliación para poder generar conjuntos de predictores utilizando como técnicas de diversificación de datos *bagging* y *class-switching*.

3.3.3 Diagrama de clases

En este apartado se presentan las principales clases y funciones que se utilizan en el programa. Todo ello, se refleja en un diagrama de clases. El objetivo es mostrar cómo está diseñado el programa y que funciones se pueden realizar en él. Las principales funcionalidades son: crear conjuntos de predictores y etiquetar ejemplos.

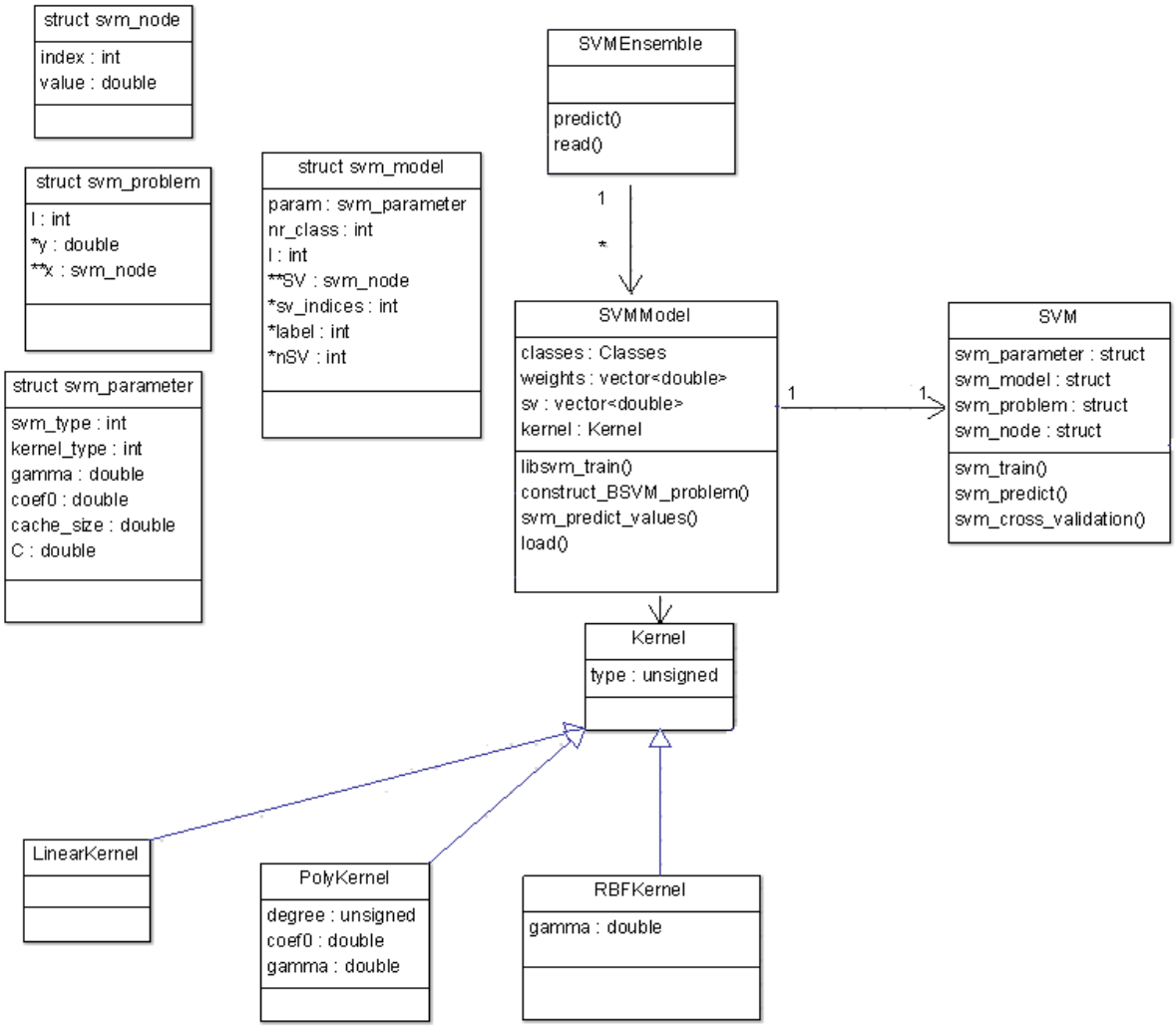

Figura 19. Diagrama de clases, clases principales para crear un conjunto de predictores y predecir con él.

Como se muestra en el diagrama de clases, el programa utiliza estructuras para almacenar datos. Se pueden observar las siguientes estructuras:

- **SVM_node:** Estructura en la que se almacenan los vectores de atributos de los ejemplos de entrenamiento y de test que se van a utilizar en el problema. En el atributo *index* de la estructura, se guarda la posición del atributo en el vector, y en el atributo *value*, el valor del atributo.

- **SVM_parameter:** En esta estructura se almacenan los hiperparámetros y parámetros necesarios, para poder configurar las máquinas de soporte vectorial. En el atributo *svm_type*, se guarda el tipo de máquina de soporte vectorial que se va a utilizar. En *kernel_type*, el tipo de núcleo. En *gamma*, el valor del hiperparámetro gamma en caso de que el núcleo lo utilice. En *coef0*, la constante de la función del núcleo en caso de utilizarla. En la variable *cache_size*, el tamaño de memoria caché (indicada en MB), que se quiere

utilizar para entrenar los predictores. En *C*, el valor del parámetro de regularización que se quiere utilizar para entrenar el predictor.

- **SVM_problem:** En esta estructura se guarda en la variable *l* el número de ejemplos de entrenamiento, en *y* las etiquetas clase, y en el atributo *x* el vector de atributos de cada ejemplo.

- **SVM_model:** En SVM_model se almacena un predictor entrenado. Para realizar esta acción, el programa guarda en *param* los hiperparámetros que se van a utilizar para el entrenamiento del predictor. En *nr_class* el número de clases que hay en el conjunto de entrenamiento. En *l* en número de vectores soporte que se han encontrado en el problema. En *SV* se almacenan todos los vectores soporte. En *sv_indices* el índice correspondiente al ejemplo (número de línea) que ocupa el vector soporte en el conjunto de entrenamiento. En *label*, la etiqueta de cada clase. Y en *nSV*, se almacena la clase de cada vector soporte.

A continuación, se van a explicar las clases que se han utilizado en el programa mostradas en el diagrama de clases.

Para crear el núcleo que se va a utilizar en las máquinas de soporte vectorial, se utilizan las clases *LinearKernel, PolyKernel, RBFKernel*. Todas ellas, heredan de la clase *Kernel.* Dependiendo del núcleo que se va a utilizar, las clases guardan como atributos, distintos hiperparámetros. En la clase *Kernel*, se almacena el tipo de núcleo que se va a utilizar en el atributo *type*. En la clase *PolyKernel*, se guarda en el atributo *degree* el grado del polinomio. En *coef0* la constante polinomial en caso de usarse. Y en *gamma* la pendiente. En la clase *RBFKernel*, la variable *gamma* guarda la anchura del núcleo, (ver Anexo A con todos los tipos de núcleo).

Para configurar una máquina de soporte vectorial se utiliza la clase *SVM.* Esta clase, permite almacenar en estructuras los valores de los hiperparámetros que utiliza la máquina de soporte vectorial (estructura *svm_parameter*), los ejemplos de entrenamiento (estructura *svm_problem*) y los vectores soporte encontrados con sus clases (estructura *svm_model*). En esta clase, podemos encontrar las funciones para realizar la validación cruzada, el entrenamiento de un predictor y la predicción de etiquetas con un predictor.

Para guardar un predictor entrenado, se utiliza la clase *SVMModel*. En esta clase, se guarda la configuración de la máquina de soporte vectorial (clase *SVM*), los vectores soporte en el atributo *sv* y su clase en el atributo *classes*. Por último, guarda el vector pesos que define el hiperplano separador en el atributo *weights*. Como funciones, podemos encontrar la función que guarda un predictor (clase *SVMModel*) en un fichero o la función de cargar un predictor a la clase *SVMModel* desde un fichero.

Por último, tenemos la clase que nos permite crear un conjunto de predictores. Esta clase es *SVMEnsemble.* En esta clase, se almacenan todos los predictores que se hayan generado (clase *SVMModel*). Respecto a las funciones, resalta la función para predecir con un conjunto de predictores (clase *SVMModel*) y la función que nos permite cargar un conjunto de predictores guardados en un fichero.

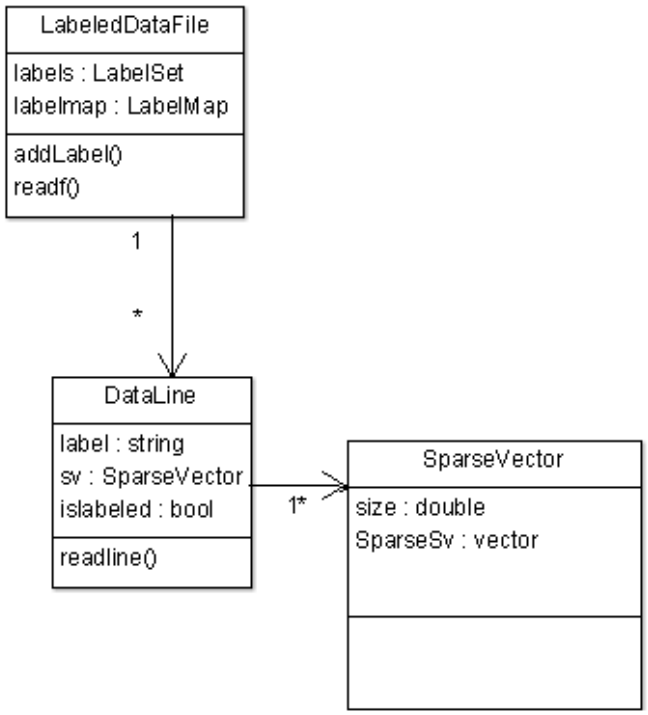

Figura 20. Diagrama de clases, clases para el trato de los datos leídos y vectores

En el diagrama de la figura 20, se observan las principales clases para poder leer y almacenar los ejemplos de un fichero, ya sean etiquetados o sin etiquetar.

Para leer un ejemplo de los ficheros que contienen el conjunto de entrenamiento y test, se utiliza la clase *DataLine.* La función *readline ()* se encarga de formatear una línea del fichero. Para formatearla separa la clase del vector de atributos. La clase se guarda en el atributo *label* y el vector de atributos en la variable *sv*. Para almacenar el vector de atributos se utiliza la clase *SparseVector*, en la que se guarda el número de atributos que tiene el vector (variable *size*), y el vector de atributos (variable *SparseSv*).

La clase *LabeledDataFile* se utiliza para almacenar todos los ejemplos del fichero leído. En el atributo *labelmap* se guarda todas las etiquetas clase con el índice del vector de atributos que le corresponde. En *labels* se almacena las etiquetas de clase que hay en el problema. Por último, almacena todas las líneas del conjunto de entrenamiento que se guardan en la clase *DataLine*. Para facilitar la obtención de los vectores de atributos y clases, se utiliza el patrón de diseño iterator.

3.3.4 Funciones utilizadas en el programa

En este apartado se van a explicar las funciones principales que se utilizan en el programa para poder generar un conjunto de SVMs y etiquetar un conjunto de ejemplos test. Se van a detallar las funciones en tablas. En dichas tablas, se mostrará el nombre de la función, los parámetros de entrada necesarios, la salida de la función y una breve descripción.

Nombre de la función	*svm_cross_validation (**const struct svm_problem** *prob, **const struct svm_parameter** *param, **int** nr_fold, **double** *target)*
Parámetros de entrada	-*prob*: Estructura que contiene los ejemplos de entrenamiento -*param*: Estructura que almacena los hiperparámetros de la máquina de soporte vectorial -*nr_fold*: Número de particiones que se van a realizar -*target*: Variable en la que se guarda la tasa de acierto obtenida en la validación del predictor
Salida	-Tasa de acierto del predictor al predecir la etiqueta clase de la partición de validación -Valores de los hiperparámetros que utilizan las máquinas de soporte vectorial
Descripción	Esta función nos permite realizar la validación cruzada sobre un conjunto de ejemplos de entrenamiento, así poder obtener el valor correcto de los hiperparámetros de las máquinas de soporte vectorial, para poder aplicarlos en el entrenamiento del predictor

Tabla 1. Función *svm_cross_validation*

Nombre de la función	*bootstrap(**char ***data**, **char ***ofname**,**char ***labels, **int** nboot)*
Parámetros de entrada	-*data*: Nombre del fichero que contiene el conjunto de entrenamiento -*ofname*: Nombre del fichero de salida -*labels*: Nombre de las clases separadas por espacio -*nboot:* Número de conjuntos a crear
Salida	-Fichero, con los conjuntos de ejemplos creados con *bagging*
Descripción	Esta función aplica la técnica de remuestreo de datos *bagging* sobre el conjunto de entrenamiento. Crea tantos conjuntos de datos como predictores tenga el colectivo

Tabla 2. Función *bootstrap*

Nombre de la función	*readBootstrapLine(**std::istream** &stream, **std::list<unsigned>** &mask, **char** delim)*
Parámetros de entrada	-*stream*: Línea del fichero que contiene el conjunto de ejemplos creado con *bootstrap* -*mask*: Contenedor para almacenar los ejemplos creados con la técnica *bagging* -*delim*. Delimitador que separa los ejemplos en el fichero de entrada
Salida	-Contenedor, con los ejemplos creados con la técnica *bagging*

Descripción	Esta función lee las líneas del fichero creado por la función *bootstrap*, almacenándolas en un contenedor para posteriormente utilizarlas.

Tabla 3. Función *readBootstrapLine*

Nombre de la función	*readline(**const std::string** &line, **int** format=0)*
Parámetros de entrada	-*line*: Línea del fichero del conjunto de entrenamiento -*format*: Formato del fichero
Salida	-Clase *DataLine* dónde almacena la etiqueta clase y el vector de atributos de cada línea leída
Descripción	Formatea una línea leída del conjunto de entrenamiento. Para ello, separa la clase del vector de atributos y lo almacena en la clase *DataLine*

Tabla 4. Función *readline*

Nombre de la función	*KernelFactory* **(*unsigned** kfun, **unsigned** degree, **double** gamma, **double** coef0)*
Parámetros de entrada	-*kfun*: Tipo de núcleo -*degree*: Valor del grado del núcleo polinomial -*gamma*: Valor de gamma (γ) -*coef0*: Valor de la constante del núcleo
Salida	-El núcleo que se va a utilizar (clase *kernel*)
Descripción	Esta función, nos permite crear el núcleo que van a utilizar las máquinas de soporte vectorial. Para crear el núcleo, se utiliza el patrón de diseño Factory

Tabla 5. Función *KernelFactory*

Nombre de la función	*construct_BSVM_problem* **(*const Kernel** *kernel, **double** pospen, **double** negpen, **double** cachesize, **const vector<const SparseVector*>** &data, **const vector<bool>** &labels, **const vector<double>** &penalties, **unsigned** trainsize)*
Parámetros de entrada	-*kernel*: El núcleo que se va a utilizar -*pospen*: Penalización en clases positivas -*negpen*: Penalización en clases negativas -*cachesize*: Tamaño de la memoria caché en MB -*data*: Vector de atributos de los ejemplos del conjunto de entrenamiento -*labels*: Etiquetas de los ejemplos, true clase positiva, false clase negativa -*penalties*: penalidad de los ejemplos -*trainsize*: El número de ejemplos que contiene el conjunto de entrenamiento
Salida	-Estructura *SVM_problem* inicializada con los valores que va a utilizar -Estructura *SVM_parameter* configurada con los hiperparámetros que se van a utilizar en el problema
Descripción	Esta función nos permite almacenar los hiperparámetros y los datos de entrenamiento para poder entrenar un

	predictor. Lo primero que realiza la función, es configurar la estructura *SVM_parameter*. El programa almacena en la estructura los valores necesarios para crear una máquina de soporte vectorial. A continuación, configura la estructura *SVM_problem* con el número de ejemplos de entrenamiento que se van a utilizar, la etiqueta clase y el vector de atributos de cada ejemplo

Tabla 6. Función *Construct_BSVM_problem*

Nombre de la función	*svm_train(**const svm_problem** *prob, **const svm_parameter** *param)*
Parámetros de entrada	-*prob*: Estructura *svm_problem,* que contiene el problema a resolver -*param*: Estructura *svm_parameter,* que contiene los hiperparámetros
Salida	-Devuelve un predictor (estructura *svm_model*)
Descripción	Esta función se encarga de entrenar un único predictor. Para ello, crea la estructura *svm_model* (predictor entrenado). En esta estructura, la función almacena el número de vectores soporte que ha encontrado de cada clase, los vectores soporte y la frontera de decisión. Para almacenar todos los vectores soporte, utiliza la estructura *svm_node*

Tabla 7. Función *svm_train*

Nombre de la función	*libsvm_train(**full_svm_problem &&**problem**))*
Parámetros de entrada	-*problem*: Contenedor en el que se almacena la estructura *svm_problem,* que contiene el problema a resolver, y la estructura *svm_parameter,* que contiene los hiperparámetros
Salida	-Devuelve el predictor almacenado en la clase *SVMModel*
Descripción	Esta función se encarga de llamar a la función *Svm_train,* pasando como argumento de entrada la estructura *svm_problem* y la estructura *svm_parameter.* Como salida, devuelve el predictor almacenado en la clase *SVMModel.* Para crear la clase *SVMModel,* almacena en los atributos de la clase, los valores de la estructura *svm_model,* que ha devuelto la función s*vm_train*

Tabla 8. Función *libsvm_train*

Nombre de la función	*ESVM-train(**char** * data, **char** *ofile, **char** * labels, **double** pospen, **double** negpen, **int** kfun, **int** svm,**int** nmodels, **char** *bfile, **double** gamma, **double** coef0, **double** degree)*
Parámetros de entrada	-*data*: Nombre del fichero del conjunto de entrenamiento -*ofile*: Nombre del fichero de salida -*labels*: Etiquetas del conjunto de entrenamiento -*pospen*: Penalización en clases positivas

	-*negpen*: Penalización en clases negativas -*kfun*: Tipo de núcleo a utilizar -*svm*: Tipo de máquina de soporte vectorial -*nmodels*: Número de predictores a crear -*bfile*: Nombre del fichero que se ha generado con *bagging* -*degree*: Valor del grado del núcleo Polinomial -*gamma*: Valor de gamma (γ) -*coef0*: Valor de la constante del núcleo
Salida	-Fichero con el conjunto de predictores
Descripción	En esta función se realiza el entrenamiento para cada predictor. Para entrenar los predictores, se utiliza los conjuntos de ejemplos generados con *bagging*. De este modo, cada predictor es entrenado con un conjunto de ejemplos diferente. Para realizar el entrenamiento de cada predictor, la función inicializa la estructura *svm_problem* y *svm_parameter* con los valores necesarios para poder entrenar el predictor. A continuación, llama a la función *ESVM-train.* Cada predictor entrenado que devuelve la función *ESVM-train*, se guarda en un contenedor

Tabla 9. Función *ESVM-train*

Nombre de la función	*svm_predict_values(**const svm_model** *model, **const svm_node** *x)*
Parámetros de entrada	-*model*: El predictor -*x*: Ejemplo a predecir
Salida	-Devuelve la clase que ha predicho el predictor
Descripción	En esta función, el predictor realiza la predicción de la etiqueta clase de un ejemplo

Tabla 10. Función *svm_predict_values*

Nombre de la función	***predict(char* data, char *datamodels, char* ofile)***
Parámetros de entrada	-*data*: Nombre del fichero que contiene el conjunto de datos a predecir la etiqueta clase -*datamodels*: Nombre del fichero con el conjunto de predictores -*ofile*: Fichero de salida
Salida	-Porcentaje de acierto obtenido en el etiquetado de los ejemplos por el predictor, en caso de que los ejemplos test estén etiquetados -Fichero con las clases predichas por el predictor de cada ejemplo.
Descripción	En esta función se predice la etiqueta clase de un conjunto de ejemplos mediante un colectivo de predictores. Para ello, en la función se predice la etiqueta clase de un ejemplo con todos los predictores. Para realizar esta acción, llama a la función *Svm_predict_values*. Se usa voto por mayoría para escoger la etiqueta predicha por todos los predictores para un mismo ejemplo

Tabla 11. Función *predict*

3.3.5 Implementación de la ampliación de la librería EnsembleSVM

Para realizar el objetivo del trabajo, se va a utilizar el mismo lenguaje de programación que se utiliza en la librería EnsembleSVM, el lenguaje C++. Se utiliza la misma estructura de programación con el fin de conseguir una funcionalidad incremental de la librería EnsembleSVM. La licencia que voy a utilizar es GNU LGPL. He elegido esta licencia para que se pueda seguir estudiando y mejorando este código. Esta licencia, permite que un usuario pueda usar y modificar el código e integrarlo como código abierto o privado en un nuevo software, (se puede ver más en detalle en el Anexo B).

Lo primero que se modifica en la librería es la función *bootstrap* para introducir remuestreo en el conjunto de datos de entrenamiento. En vez de crear conjuntos del tamaño del fichero con los datos de partida, se podrá indicar como parámetro de entrada, el porcentaje de datos del total de ejemplos que se van utilizar. Se crea un fichero de salida con los nuevos conjuntos de datos. Como pasa en *class-switching*, se crean tantos conjuntos de datos como predictores se indique.

Nombre de la función	*bootstrap_modified(**char** *data, **char** * ofname,**char** *labels, **int** nboot, double rem)*
Parámetros de entrada	-*data*: Nombre del fichero que contiene el conjunto de entrenamiento -*ofname*: Nombre del fichero de salida -*labels*: Nombre de las clases separadas por espacio -*nboot*: Número de conjuntos a crear -*rem*: Remuestreo en el conjunto de entrenamiento
Salida	-Fichero con los conjuntos de ejemplos creados con *bagging_modified*
Descripción	Esta función aplica la técnica de remuestreo de datos *bagging* sobre el conjunto de entrenamiento. Crea tantos conjuntos de datos como predictores tenga el colectivo. Los conjunto de ejemplos se crean del tamaño indicado por el parámetro de entrada.

Tabla 12. Función *bootstrap_modified*

Para extender la librería EnsembleSVM, se crea una nueva función llamada *class-switching*. Esta función será la encargada de crear nuevos conjuntos de ejemplos para realizar el cambio de clase. Para ello, habrá que indicar a la función por parámetros de entrada, el nombre del conjunto de entrenamiento, el número de predictores que va a tener el conjunto y el porcentaje de datos al que se le va a cambiar de clase. Una vez indicados los parámetros, la función creará tantos conjuntos de ejemplos como predictores se le haya indicado. Se genera como salida un nuevo fichero de texto con los conjuntos creados.

La estructura del fichero creado es la siguiente:

- El fichero tiene tantas líneas como predictores (SVMs) se le indique.
- Cada línea se compone por permutaciones sin repetición. Cada valor de la permutación, es la posición del ejemplo (índice) correspondiente al fichero que contiene los ejemplos de entrenamiento.
- El tamaño de cada línea, corresponde al porcentaje de remuestreo del conjunto de entrenamiento que se indique en el problema.

- Los primeros índices por línea, serán los datos a los que se les cambie la clase, por ejemplo:
  Si tenemos 3, 6, 5, 4 y queremos cambiar la clase al 50% de los ejemplos, se le cambia la clase al 3 y al 6, los demás siguen teniendo la misma clase.

Nombre de la función	*class-switching(**char** *data, **char** * ofname,**char** *labels, **int** nsvm, **double** class-switching)*
Parámetros de entrada	*-data*: Nombre del fichero que contiene el conjunto de entrenamiento *-ofname*: Nombre del fichero de salida *-labels*: Nombre de las clases separadas por espacio *-nsvm*: Número de conjuntos a crear *-class-switching*: porcentaje de cambio de clase
Salida	-Fichero con los conjuntos de ejemplos creados con *class-switching*
Descripción	Esta función aplica la técnica de remuestreo de datos *class_switching* sobre el conjunto de entrenamiento. Crea tantos conjuntos de datos como predictores tenga el colectivo

Tabla 13 Función *class-switching*

Una vez realizados los cambios anteriormente descritos. Se crea una nueva función llamada *ReadClass-switching*. Está función se encargará de leer el fichero creado con la función *class-switching* y de almacenar todos los conjuntos de ejemplos que contiene el fichero en un contenedor.

Nombre de la función	*ReadClass-switching(**std::istream** &stream, **std::list<unsigned>** &mask, **char** delim)*
Parámetros de entrada	*-stream*: línea del fichero que contiene el conjunto de *class-switching* *-mask*: Contenedor para almacenar los ejemplos a los que se les va a realizar el cambio de clase *-delim*. Delimitador que separa los ejemplos en el fichero *class-switching*
Salida	-Contenedor con los ejemplos a cambiar la clase almacenados
Descripción	Esta función lee las líneas del fichero creado por la función *class-switching*, almacenándolas en un contenedor para posteriormente utilizarlas

Tabla 14. Función *ReadClass-switching*

Por último, se modifica la función principal del programa *ESVM-train* para poder utilizar *bagging*, *class-switching* y remuestreo. Para ello, se ha añadido dos parámetros nuevos de entrada. En el parámetro entrada "*class-switching"* se indica el nombre del fichero que contiene los conjuntos de ejemplos creados con la función *class-switching*, y en el parámetro "*rem"* se indica el remuestreo de datos. Si no se indican los nuevos parámetros, el programa tendrá la misma funcionalidad que sin la ampliación de la librería. Si se indican los nuevos parámetros, se realizará el cambio de clase y el remuestreo de datos. Para realizar el entrenamiento de cada predictor, el programa lee el

conjunto de ejemplos que le corresponde a ese predictor del fichero *bagging* y *class-switching*. Cambia la clase a los ejemplos del conjunto de *bagging* que correspondan. Una vez realizado el cambio de clase, se procede a entrenar el predictor. Cada predictor se entrena con su conjunto de ejemplos correspondiente. El predictor que se obtiene, se añade al conjunto de predictores que se devolverá como resultado en un fichero.

La implementación de c*lass-switching* se basa en el siguiente algoritmo:

for t = 1:T
surrogate_training_data = boostrap_sample(class_switching(training_data)),
h_t*= build_SVM_from (surrogate_training_data)*

Siendo *T* el número total de predictores a crear y $h_t$el predictor creado mediante la máquina de soporte vectorial a partir del conjunto de datos.

Nombre de la función	*ESVM-train_modified(**char** * data, **char** *ofile, **char** * labels, **double** pospen, **double** negpen, **int** kfun, **int** svm,**int** nmodels, **char** *bfile, **double** gamma, **double** coef0, **double** degree,**double** rem, **double** class-switching, **char** cfile)*
Parámetros de entrada	-*data*: Nombre del fichero del conjunto de entrenamiento -*ofile*: Nombre del fichero de salida -*labels*: Etiquetas del conjunto de entrenamiento -*pospen*: Penalización en clases positivas -*negpen*: Penalización en clases negativas -*kfun*: Tipo de núcleo a utilizar -*svm*: Tipo de máquina de soporte vectorial -*nmodels*: Número de predictores a crear -*bfile*: Nombre del fichero que se ha generado con *bagging* -*degree*: Valor del grado del núcleo Polinomial. -*gamma*: Valor de gamma (γ) -*coef0*: Valor de la constante del núcleo -*rem*: Porcentaje de remuestreo en el conjunto de entrenamiento -*class-switching*: Porcentaje de cambio de clase en el conjunto de entrenamiento -*cfile*: nombre del fichero *class-switching*
Salida	-Fichero con el conjunto de predictores
Descripción	En esta función, se realiza el entrenamiento para cada predictor. Para entrenar los predictores, se utiliza los conjuntos de ejemplos generados con *class-switching* a partir de los conjuntos generados con *bagging*. De este modo, cada predictor es entrenado con un conjunto de ejemplos diferente. Para realizar el entrenamiento de cada predictor, la función inicializa la estructura *svm_problem* y *svm_parameter,* con los valores necesarios para poder entrenar un predictor. A continuación, llama a la función *ESVM-train.* Cada predictor entrenado que devuelve la función *ESVM-train,* se guarda en un contenedor

Tabla 15. Función *ESVM-train_modified*

3.3.6 Diagramas de secuencia

En este apartado se va a explicar la interacción entre las clases del programa. Para ello, se van a representar en diferentes diagramas de secuencia las principales funcionalidades del programa. Se mostrarán las principales clases y funciones para entrenar un conjunto de predictores y predecir un conjunto de test mediante los predictores.

- **Entrenamiento de un conjunto de predictores**

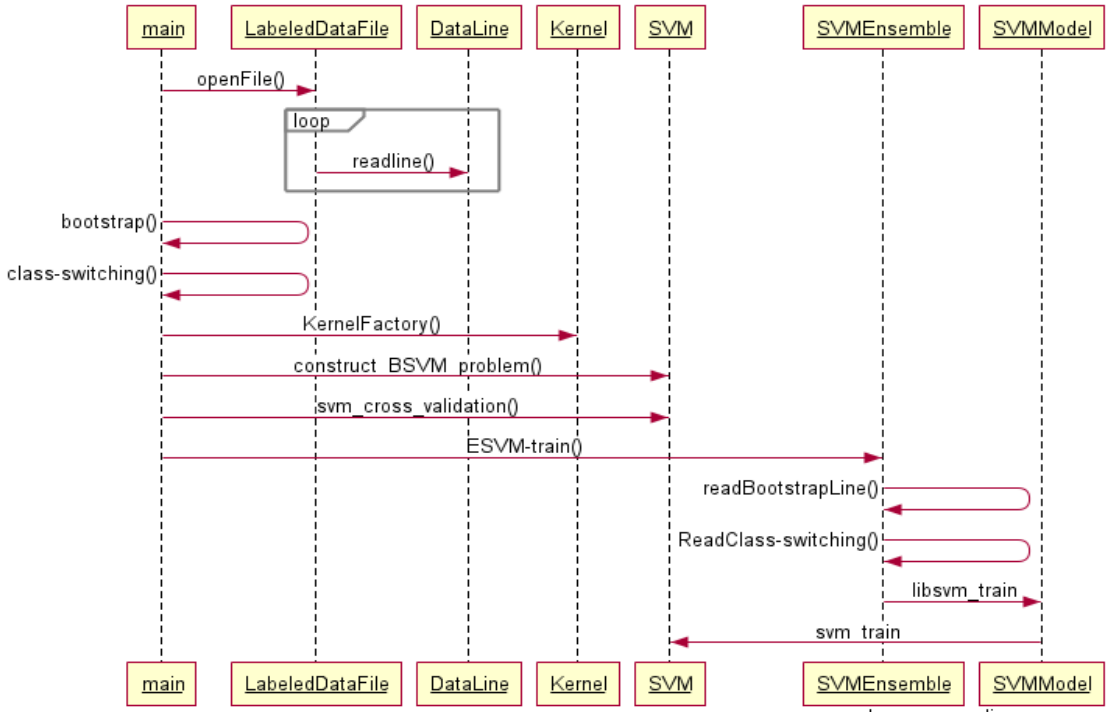

Figura 21. Diagrama de secuencia de entrenamiento de un conjunto de predictores

- **Predicción mediante un conjunto de predictores**

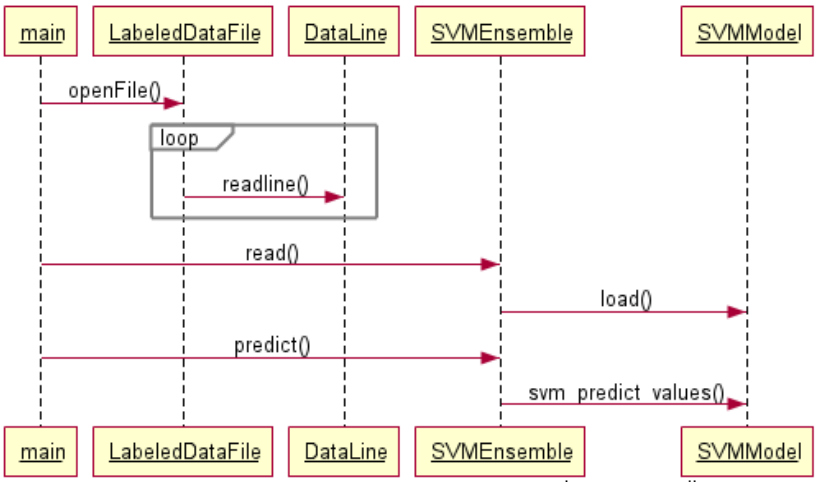

Figura 22. Diagrama de secuencia de una predicción mediante un conjunto de predictores

4 Pruebas y resultados

Las pruebas consistirán en evaluar las tasas de error obtenidas con predictores al predecir las etiquetas clase de diferentes conjuntos de datos. También se medirán los tiempos de entrenamiento de diferentes predictores para poder comparar las librerías utilizadas en este trabajo. Los conjuntos de datos que se van a utilizar en los problemas, se extraerán del repositorio *UCI* (Lichman, 2013), menos el conjunto de datos *Ijcnn* (Prokhorov, 2001), que fue extraído del repositorio LIBSVM (Chang & Lin, 2011). Estos conjuntos se dividen en dos ficheros, un fichero para realizar el entrenamiento de los predictores, y otro fichero que se utilizará para realizar las predicciones de la etiqueta clase mediante los predictores.

Como algoritmo de aprendizaje automático base se va a utilizar las máquinas de soporte vectorial. El núcleo que se va utilizar en las máquinas de soporte vectorial, es el núcleo RBF. Se ha escogido este núcleo porque se van a resolver problemas que requieren resolverlos mediante un modelo no lineal. Para configurar los hiperparámetros (c y γ) que van a utilizar las máquinas de soporte vectorial, se va a utilizar la validación cruzada con 10 particiones.

Lo primero que se realizará en las pruebas será predecir las etiquetas clases con un predictor. Para obtener el predictor se utilizará la librería LIBSVM. Se medirá el tiempo de entrenamiento del predictor para cada conjunto de entrenamiento utilizado.

A continuación, se predecirán las etiquetas clase de los datos con un conjunto de predictores. Los conjuntos de predictores se entrenarán utilizando la librería EnsembleSVM y la ampliación de EnsembleSVM. Todos los conjuntos de predictores se formarán con 501 predictores a excepción de los conjuntos que utilicen para entrenar los conjuntos de datos *Adult* e *Ijcnn1*. Esto es debido a que estos dos conjuntos son de gran tamaño y en las pruebas que hicieron de la librería EnsembleSVM utilizaron 100 SVMs (Claesen, 2014). Como técnicas de remuestreo se utilizarán *bagging* y *class-switching.*

En la técnica *bagging* se utilizarán diferentes remuestreos en el conjunto de entrenamiento. Los porcentajes que se utilizarán son del 10%, 20%,50%, 80%, 100% y 120% del total de ejemplos que contenga el conjunto.

En la técnica *class-switching* se procederá a realizar diferentes porcentajes de cambio de clase. Estos porcentajes son del 10%, 20%, 30%, 40%, y del 50% del total de ejemplos del conjunto.

Se medirá el tiempo de entrenamiento de los conjuntos de predictores para cada conjunto de datos utilizado.

Las pruebas se realizarán con un ordenador que presenta las siguientes características:

- Procesador AMD Athlon X4 750 K Quad Core de 3.40 GHz
- Memoria RAM de 8.00 GB

4.1 Conjunto de datos utilizados

En la siguiente tabla se muestran las características de los conjuntos de ejemplos utilizados. La tabla contiene el número de ejemplos, el número de atributos, el número de ejemplos que se van a utilizar en el conjunto de entrenamiento y test, el número de clases, y el valor de los hiperparámetros C y γ.

Conjunto de entrenamiento	*Ejemplos*	*Atributos*	*Train*	*Test*	*Clases*	*C*	*γ*
Pima Indian Diabetes	**768**	**8**	**538**	**230**	**2**	**8**	**3,05176e-05**
Australian Credit Approval	**690**	**14**	**493**	**207**	**2**	**512**	**3,05176e-05**
Liver Disorders	**345**	**7**	**242**	**103**	**2**	**8192**	**0,03125**
German	**1000**	**20**	**750**	**250**	**2**	**32768**	**3,05176e-05**
Adult	**48842**	**123**	**32.561**	**16.281**	**2**	**1**	**1**
Ijcnn1	**141691**	**22**	**49.990**	**91.701**	**2**	**1**	**32**

Tabla 16. Conjunto de datos UCI

En la tabla 16 se puede observar que los conjuntos de datos de más tamaño son *Adult* e *Ijcnn1*. El más pequeño es *Liver Disorders*. Todos ellos, se componen de dos clases. Para el conjunto de entrenamiento se han utilizado más o menos el 70% del total de ejemplos y el resto de ejemplos para el conjunto test menos en el conjunto de datos *Ijcnn1*, que se han utilizado el 30% de los ejemplos para el entrenamiento.

4.2 Resultados con un predictor utilizando LIBSVM

A continuación, se muestran las tasas de error de las etiquetas clase predichas por un único predictor para los conjuntos de ejemplos que se van a utilizar en las pruebas descritas en el apartado anterior. También se muestra el tiempo que ha tardado el programa en entrenar un predictor para cada conjunto de entrenamiento. Las máquinas de soporte vectorial, se han configurado con los hiperparámetros que se muestran en la Tabla 16.

Conjunto de ejemplos de entrenamiento	*Tasa de error*	*Tiempo de entrenamiento del predictor*
Pima Indian Diabetes	**27,83%**	**0,054 s**

Australian Credit Approval	**25,61%**	**0,056 s**
Liver Disorders	**37,07%**	**0,049 s**
German	**24,00%**	**0,139 s**
Adult	**19,33%**	**667,216 s**
Ijcnn1	**01,33%**	**56,674 s**

Tabla 17. Tasa de acierto con una SVM

Se puede apreciar que con un predictor el tiempo de entrenamiento es mayor en los conjuntos de datos *Adult* e *Ijcnn*. Se debe a que el volumen de ejemplos que contienen los dos conjuntos es muy superior a los otros conjuntos de ejemplos. En el caso del conjunto de ejemplos *Adult* al contener 123 atributos, encontrar un hiperplano separador en el problema es más costoso que en los otros conjuntos de ejemplos. Con lo cual, entrenar el predictor con este conjunto de ejemplos conlleva más tiempo que entrenar el predictor con los demás conjuntos de ejemplos. También, se puede observar cómo va mejorando la tasa de error a medida que los conjuntos de ejemplos son más grandes. Esto sucede porque el predictor utiliza más ejemplos para entrenar, de esta manera obtiene más información para encontrar patrones que relacionen el vector de características con las clases de los ejemplos.

4.3 Resultados de conjuntos de predictores utilizando bagging como técnica de remuestreo

En este apartado se va a ilustrar la tasa de error de la predicción de la etiqueta clase con un conjunto de predictores que utilizan *bagging* como técnica de remuestreo de datos. Se utilizan diferentes porcentajes de remuestreo en los datos de entrenamiento para obtener el conjunto de predictores entrenados. También se muestra el tiempo de entrenamiento en segundos del conjunto de predictores que ha tardado el programa en entrenarlos. Los porcentajes en negrita, son las mejores tasas de error que han obtenido el conjunto de predictores al etiquetar el conjunto de datos.

	Remuestreo					
Conjunto de entrenamiento	***10%***	***20%***	***50%***	***80%***	***100%***	***120%***
Pima Indian Diabetes	30,70% 0,436 s	29,74% 0,895 s	28,39% 2,397 s	28,09% 3,949 s	28,05% 5,627 s	**28,02%** 7,633 s
Australian Credit Approval	37,06% 0,495 s	33,53% 0,986 s	29,76% 2,878 s	29,33% 4,689 s	26,63% 6,602 s	**25,83%** 9,357 s
Liver Disorders	39,27% 0,199 s	36,67% 0,406 s	36,73% 0,984 s	35,44% 1,812 s	35,18% 2,228 s	**34,82%** 2,614 s
German	32,75% 1,058 s	29,66% 2,309 s	27,97% 6,946 s	25,84% 16,046 s	24,43% 24,084 s	**24,08%** 29,761 s
Adult	20,70% 108,3 s	20,51% 650,435 s	20,48% 5496,547 s	20,30% 12477,853 s	20,14% 18949,465 s	**20,09%** 24673,384s
Ijcnn1	02,90% 33,73 s	02,48% 106,671 s	02,05% 525,165 s	01,80% 1210,202 s	01,76% 1842,678 s	**01,71%** 2043,823 s

Tabla 18. Tasa de error con un conjunto de predictores que han utilizado *bagging* como técnica de remuestreo en los ejemplos de entrenamiento

En la Tabla 18 se puede comprobar que cuanto menor es el remuestreo utilizado para entrenar el conjunto de predictores, peor tasa de error se obtiene. Se debe a que el entrenamiento del conjunto de predictores se realiza con menos ejemplos, por tanto, la probabilidad de fallar es mayor ya que, al utilizar pocos ejemplos el predictor no contempla todos los casos (tiene falta de información).

Predecir con un conjunto de predictores tiene la ventaja de que no empeora la tasa de error significativamente, es decir, la tasa de error se asemeja para todas las predicciones mediante predictores entrenados a partir de conjuntos de datos con diferentes remuestreos. Cuantos más ejemplos formen los conjuntos de entrenamiento, más se iguala la tasa de error en la predicción mediante los diferentes conjuntos de predictores.

En el conjunto de ejemplos más voluminoso (*Ijcnnl*), la tasa de error que obtiene el conjunto de predictores entrenados con un remuestreo del 10% es de 02,90%, mientras que la tasa de error de la predicción con el 100% de remuestreo en el conjunto de entrenamiento es de 01.71%. Lo que indica que las dos tasas de error son buenas con la diferencia de que el entrenamiento con el 10% de remuestreo ha sido de 33,73 segundos, mientras que utilizando el 100% de los ejemplos, el tiempo de entrenamiento ha sido de 1842,678 segundos.

4.4 Resultados de conjuntos de predictores utilizando bagging y Class Switching como técnicas de remuestreo

En las siguientes tablas se van a mostrar las tasas de error de la predicción de la etiqueta clase por un conjunto de predictores y su tiempo de entrenamiento. Para realizar el entrenamiento de los predictores, se han utilizado *bagging* y *class-switching* como técnicas de remuestreo de datos. Para cada porcentaje de remuestreo en el conjunto de entrenamiento se aplican todos los porcentajes de cambio de clase explicados. Debajo de las tasas de error, se muestra el tiempo de entrenamiento en segundos. Los porcentajes en negrita, son las mejores tasas de error que han obtenido el conjunto de predictores al etiquetar el conjunto de datos. La última columna corresponde a los tiempos de entrenamiento y a las tasas de error de los predictores que han utilizado *bagging* únicamente como técnica de remuestreo.

- **Remuestreo 10%**

	Cambio de clase					
Conjunto de entrenamiento	**10%**	**20%**	**30%**	**40%**	**50%**	***Bagging***
Pima Indian Diabetes	**30.64%** 0.446 s	31.09% 0.453 s	31.16% 0.448 s	31.59% 0.452 s	31.60% 0.451 s	30.70% 0.436 s
Australian Credit Approval	**29.90%** 0.507 s	39.02% 0.512 s	37.13% 0.510 s	37.31% 0.509 s	37.66% 0.513 s	37.06% 0.495 s
Liver Disorders	**39.20%** 0.201 s	39.46% 0.204 s	39.67% 0.202 s	39.79% 0.201 s	39.83% 0.205 s	39.27% 0.199 s
German	**31.15%** 1.064 s	31.49% 1.068 s	31.52% 1.063 s	32.59% 1.071 s	32.80% 1.070 s	32.75% 1.058 s
Adult	21.33% 108.321 s	21.41% 108.368 s	21.69% 108.404 s	21.78% 108.310 s	21.97% 108.423 s	**20.70%** 108.3 s
Ijcnn1	03.32% 33.751 s	03.63% 33.738 s	03.98% 33.742 s	04.21% 33.732 s	04.72% 33.745 s	**02.90%** 33.73 s

Tabla 19. Tasa de error con un conjunto de predictores que han utilizado *bagging* y Class Switching como técnica de remuestreo en los ejemplos de entrenamiento. Remuestreo del 10% en el conjunto de entrenamiento

En la tabla anterior se observa que cuanto menor es el conjunto de entrenamiento, con un 10% de inyección de ruido, la tasa de error de la predicción de clases es mejor que la tasa de error que se obtiene con el conjunto de predictores que solo utiliza *bagging*. En cambio, cuanto mayor es el conjunto de ejemplos de entrenamiento (*Adult e Ijcnn1*), peor

es la tasa de error con cambio de clase. Nunca mejora la tasa de error de la predicción con un conjunto de predictores que únicamente utiliza *bagging* como técnica de remuestreo.

Con un remuestreo del 10% en el conjunto de entrenamiento, la tasa de error de las predicciones de etiquetas clase se asemeja en todos los casos. El tiempo de entrenamiento es similar ya que varía por ruido (procesos que se estén ejecutando en el ordenador y afecte a los recursos en el momento que se ha lanzado la prueba). Es muy parecido al tiempo obtenido en el entrenamiento de los predictores que solo utilizan *bagging*.

- **Remuestreo 20%**

	Cambio de clase					
Conjunto de entrenamiento	**10%**	**20%**	**30%**	**40%**	**50%**	***Bagging***
Pima Indian Diabetes	**29.53%** 0.910 s	30.04% 0.916 s	30.45% 0.912 s	30.52% 0.914 s	30.66% 0.912 s	29.74% 0.895 s
Australian Credit Approval	**33.40%** 0.998 s	34.78% 1.004 s	35.08% 1.002 s	35.64% 1.008 s	36.38% 1.005 s	33.53% 0.986 s
Liver Disorders	37.07% 0.422 s	37.94% 0.429 s	39.39% 0.420 s	39.74% 0.425 s	39.06% 0.424 s	**36.67%** 0.406 s
German	**29.53%** 2.330 s	30.11% 2.323 s	30.28% 2.331 s	30.91% 2.328 s	31.49% 2.320 s	29.66% 2.309 s
Adult	20.90% 650.452 s	20.97% 650.445 s	21.00% 650.471 s	21.06% 650.489 s	21.13% 650.479 s	**20.51%** 650.435 s
Ijcnn1	02.71% 106.688 s	02.98% 106.679 s	03.29% 106.692 s	03.88% 106.699 s	04.66% 106.676 s	**02.48%** 106.671 s

Tabla 20. Tasa de error con un conjunto de predictores que han utilizado *bagging* y Class Switching como técnica de remuestreo en los ejemplos de entrenamiento. Remuestreo del 20% en el conjunto de entrenamiento

Con un remuestreo del 20% en el conjunto de ejemplos de entrenamiento, la tasa de error en la predicción sigue asemejándose en todos los casos. Se aprecia que cuanto más cambio de clase hay en el conjunto de entrenamiento, peor es la tasa de error del predictor.

- **Remuestreo 50%**

	Cambio de clase					
Conjunto de entrenamiento	**10%**	**20%**	**30%**	**40%**	**50%**	***Bagging***
Pima Indian Diabetes	28.62% 2.421 s	28.82% 2.428 s	29.30% 2.414 s	29.94% 2.422 s	31.20% 2.425 s	**28.39%** 2.397 s
Australian Credit Approval	31.10% 2.890 s	32.52% 2.892 s	34.19% 2.887 s	35.43% 2.895 s	37.74% 2.893 s	**29.76%** 2.878 s
Liver Disorders	36.99% 0.995 s	37.84% 0.998 s	29.44% 0.991 s	40.18% 1.001 s	42.51% 0.996 s	**36.73%** 0.984 s
German	28.48% 6.967 s	29.91% 6.959 s	30.20% 6.970 s	30.62% 6.958 s	32.53% 6.961 s	**27.97%** 6.946 s
Adult	20.61% 5496.589 s	20.76% 5496.562 s	21.13% 5496.591 s	21.38% 5496.583 s	21.54% 5496.556 s	**20.48%** 5496.547 s
Ijcnn1	02.65% 525.191 s	02.90% 525.174 s	03.11% 525.185 s	03.42% 525.169 s	03.98% 525.178 s	**02.05%** 525.165 s

Tabla 21. Tasa de error con un conjunto de predictores que han utilizado *bagging* y Class Switching como técnica de remuestreo en los ejemplos de entrenamiento. Remuestreo del 50% en el conjunto de entrenamiento

Con el 50% de remuestreo en el conjunto de entrenamiento, las tasas de error de las predicciones no mejoran en ningún caso a la tasa de error del conjunto de predictores que solo utiliza *bagging* para el entrenamiento de los predictores.

Como se puede observar en la Tabla 21, la tasa de error de los conjuntos de predictores que utilizan conjuntos de ejemplos pequeños para el entrenamiento, es considerablemente peor cuanto mayor inyección de ruido tiene el conjunto de entrenamiento. En cambio, en los conjuntos de entrenamiento grandes, las tasas de error se asemejan, aunque empeoran ligeramente cuanto mayor ruido tienen los conjuntos de entrenamiento.

- **Remuestreo 80%**

	Cambio de clase					
Conjunto de entrenamiento	**10%**	**20%**	**30%**	**40%**	**50%**	***Bagging***
Pima Indian Diabetes	28.67% 3.958 s	29.63% 3.961 s	31.35% 3.967 s	33.82% 3.960 s	39.07% 3.959 s	**28.09%** 3.949 s

Australian Credit Approval	30.56% 4.700 s	34.57% 4.702 s	37.06% 4.697 s	41.76% 4.699 s	47.13% 4.707 s	**29.33%** 4.689 s
Liver Disorders	36.62% 1.821 s	37.21% 1.827 s	39.88% 1.823 s	42.31% 1.830 s	44.96% 1.828 s	**35.44%** 1.812 s
German	26.08% 16.065	27.19% 16.059	30.69% 16.063	32.60% 16.059	37.13% 16.068	**25.84%** 16.046 s
Adult	20.57% 12477.860 s	20.68% 12477.899 s	21.25% 12477.856 s	21.61% 12477.872 s	21.80% 12477.887 s	**20.30%** 12477.853 s
Ijcnn1	02.54% 1210.208 s	02.98% 1210.224 s	03.25% 1210.217 s	04.06% 1210.205 s	04.98% 1210.231 s	**01.80%** 1210.202 s

Tabla 22. Tasa de error con un conjunto de predictores que han utilizado *bagging* y Class Switching como técnica de remuestreo en los ejemplos de entrenamiento. Remuestreo del 80% en el conjunto de entrenamiento

Con el 80% de remuestreo en el conjunto de entrenamiento, se observa un comportamiento parecido en las tasas de error que con el 50% de remuestreo. En los predictores que utilizan conjuntos de entrenamiento más pequeños, la tasa de error empeora considerablemente cuanto mayor cambio de clase se aplique en los conjuntos de entrenamiento. En cambio, los conjuntos de entrenamiento grandes como *Adult* e *Ijcnn*, la tasa de error empeora, pero se asemeja en todos los casos.

- **Remuestreo 100%**

	Cambio de clase					
Conjunto de entrenamiento	**10%**	**20%**	**30%**	**40%**	**50%**	***Bagging***
Pima Indian Diabetes	28.49% 5.635 s	30.48% 5.643 s	32.96% 5.640 s	37.91% 5.644 s	49.06% 5.636 s	**28.05%** 5.627 s
Australian Credit Approval	30.28% 6.620 s	33.96% 6.627 s	38.81% 6.619 s	43.84% 6.625 s	49.60% 6.621 s	**26.63%** 6.602 s
Liver Disorders	36.56% 2.241 s	39.23% 2.246 s	42.03% 2.238 s	46.28% 2.240 s	50.18% 2.244 s	**35.18%** 2.228 s
German	25.19% 24.100	26.12% 24.104	21.06% 24.107	36.68% 24.096	43.61% 24.099	**24.43%** 24.084 s

Adult	20.50% 18949.479	20.64% 18949.487	21.40% 18949.472	21.77% 18949.490	21.96% 18949.474	**20.14%** 18949.465 s
Ijcnn1	02.47% 1842.690 s	03.12% 1842.700 s	03.76% 1842.692s	04.70% 1842.684s	05.42% 1842.693s	**01.76%** 1842.678 s

Tabla 23. Tasa de error con un conjunto de predictores que han utilizado *bagging* y Class Switching como técnica de remuestreo en los ejemplos de entrenamiento. Remuestreo del 100% en el conjunto de entrenamiento

Con remuestreo del 100% en el conjunto de entrenamiento, ocurre lo mismo que con el 80% de remuestreo. En los predictores que utilizan conjuntos de entrenamiento más pequeños, la tasa de error empeora considerablemente cuanto mayor cambio de clase se aplique en el conjunto de entrenamiento. En cambio, los conjuntos de entrenamiento grandes como *Adult e Ijcnn*, la tasa de error empeora ligeramente cuanto más cambio de clase se aplique.

- **Remuestreo 120%**

Cambio de clase

Conjunto de entrenamiento	**10%**	**20%**	**30%**	**40%**	**50%**	***Bagging***
Pima Indian Diabetes	28.48% 7.652 s	30.16% 7.660 s	33.01% 7.658 s	39.57% 7.649 s	49.63% 7.650 s	**28.02%** 7.633 s
Australian Credit Approval	29.73% 9.370 s	33.86% 9.363 s	38.80% 9.368 s	44.07% 9.371 s	50.11% 9.364 s	**25.83%** 9.357 s
Liver Disorders	36.53% 2.626 s	39.59% 2.621 s	42.28% 2.630 s	46.10% 2.629 s	50.16% 2.622 s	**34.82%** 2.614 s
German	25.08% 29.781 s	26.24% 29.776 s	32.11% 29.779 s	36.79% 29.787 s	43.63% 29.783 s	**24.08%** 29.761 s
Adult	20.45% 24673.393 s	20.70% 24673.402 s	21.58% 24673.397 s	22.00% 24673.389 s	22.11% 24673.395 s	**20.09%** 24673.384s
Ijcnn1	02.38% 2043.851 s	03.07% 2043.847 s	03.98% 2043.832 s	04.77% 2043.844 s	05.82% 2043.830 s	**01.71%** 2043.823 s

Tabla 24. Tasa de error con un conjunto de predictores que han utilizado *bagging* y Class Switching como técnica de remuestreo en los ejemplos de entrenamiento. Remuestreo del 120% en el conjunto de entrenamiento

Con el 120% de remuestreo ocurre lo mismo que con el 100%. Cuanto mayor cambio de clase se aplique en los conjuntos de entrenamiento pequeños, peor tasa de error se obtiene en la predicción. En cambio, los conjuntos de entrenamiento grandes como *Adult e Ijcnn*, la tasa de error empeora, pero se asemeja en todos los porcentajes de cambio de clase utilizados en los conjuntos de entrenamiento.

4.4.1 Comparativa de tiempos de entrenamiento de conjuntos de predictores

En la siguiente gráfica se va a representar los tiempos de entrenamiento de los conjuntos de predictores que se han entrenado a partir de diferentes conjuntos de datos. Para medir los tiempos de entrenamiento del conjunto de predictores se han utilizado diferentes porcentajes de remuestreo en los conjuntos de datos.

En el eje vertical de la gráfica, se puede observar el tiempo que el programa ha tardado en entrenar el conjunto de predictores. En el eje horizontal, se observan los diferentes remuestreos utilizados en el entrenamiento de los predictores.

Figura 23. Tiempo de entrenamiento de un conjunto de predictores respecto a la tasa de remuestreo en los conjuntos de entrenamiento

En la gráfica se observa que el tiempo de entrenamiento de los predictores varía con los diferentes porcentajes de remuestreo. Cuantos más ejemplos se utilicen en los conjuntos de entrenamiento, más tarda el programa en entrenar el conjunto de predictores.

En cuanto a las tasas de error de las predicciones, con menos remuestreo de datos las tasas de error empeoran, aunque se asemejan a las predicciones de los predictores que han utilizado el 100% de los datos en el entrenamiento. El tiempo de entrenamiento de los predictores con conjuntos de datos con remuestreo bajo, es significativamente inferior al tiempo de entrenamiento con el 100% de remuestreo en los datos. Esto ocurre porque al predecir las etiquetas clase de los ejemplos de test con un conjunto de predictores, penaliza menos la mala predicción que al utilizar un único predictor, ya que proporcionan predicciones globales más robustas.

Respecto al tiempo de entrenamiento, entrenar un predictor con la librería LIBSVM con el 100% de remuestreo del conjunto *ijcnn1* ha tardado 56,674 segundos. Mientras que entrenar 100 predictores con el mismo conjunto de datos con la librería EnsembleSVM ha tardado 1842,678 segundos. Con lo cual, el tiempo de entrenamiento de un predictor sería de 18,42 segundos aproximadamente. El tiempo de entrenamiento mejora significativamente si se utiliza la librería EnsembleSVM, ya que se ha implementado para que se pueda ejecutar el programa con multi-hilo.

5 Conclusiones y trabajo futuro

Como resultado de este trabajo se ha realizado una extensión de la librería EnsembleSVM. Esta librería permite generar un conjunto de máquinas de vectores soporte utilizando remuestreo de datos. La extensión realizada consiste en incorporar la posibilidad de generar conjuntos de SVMs mediante *class-switching* (Breiman, Randomizing outputs to increase prediction accuracy, 2000) y (Martínez-Muñoz & Suárez, 2005).

Como conclusión del estudio realizado se puede destacar, que, en general utilizar conjuntos de SVMs, para problemas de clasificación binaria no mejora la tasa de error respecto a SVMs individuales. Esta observación es coherente con las realizadas en la literatura, pero no había sido analizado para conjuntos de predictores generados mediante *class-switching*. A la vista de los resultados, parece que las SVMs son muy sensibles a las perturbaciones en los datos, sean estas las asociadas al remuestreo, o a la inyección de ruido en las etiquetas de clase. Esto hace que sea difícil generar conjuntos de SVMs complementarias para las cuales los mecanismos de combinación de decisión compensen los errores de predicción. No obstante, este resultado, generar conjuntos de SVMs a partir de muestras *bootstrap* mucho más pequeñas que el conjunto individual obtiene resultados razonables con un coste computacional mucho menor que *bagging*. En conjuntos de datos grandes (por ejemplo, *Adult* o *Ijcnn1*), si es aceptable una leve pérdida de calidad en la tasa de acierto, utilizar conjuntos de SVM puede ser ventajoso desde el punto de vista de coste computacional.

Siguiendo la línea de investigación y desarrollo de este trabajo, sería interesante incorporar *class-switching* a otras librerías estándar que utilicen diferentes algoritmos de aprendizaje automático, como árboles de decisión o redes neuronales, en los que la técnica ha demostrado ser más eficaz que en las SVM. De esta manera, se podría mejorar los sistemas de predicciones mediante conjuntos de predictores en más campos del aprendizaje automático.

6 Referencias

Bishop, C. (2013). *Pattern recognition and machine learning* (1 ed.). New York: Springer.

Breiman, L. (1996). Bagging predictors. *Machine Learning, 24*(2), 123-140.

Breiman, L. (2000). Randomizing outputs to increase prediction accuracy. *Machine Learning, 40*(3), 229-242.

Chang, C.-C., & Lin, C.-J. (2011). LIBSVM : a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology, 2*(3), 1-27. Obtenido de http://www.csie.ntu.edu.tw/~cjlin/libsvm

Claesen, M. D. (2014). EnsembleSVM: a library for ensemble learning using support vector machines. *Journal of Machine Learning Research*, 141-145.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning, 20*(3), 273-297.

Dietterich, T. (2000). Ensemble Methods in Machine Learning. *Multiple Classifier Systems*, 1-15. Obtenido de http://dx.doi.org/10.1007/3-540-45014-9_1

Duda, R., Hart, P., & Stork, D. (2012). *Pattern Classification* (1 ed.). New York: John Wiley & Sons.

Efron, B., & Tibshirani, R. (2000). *An introduction to the bootstrap.* Chapman & Hall/CRC: Boca Raton, Florida.

Freund, Y., & Schapire, R. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences, 55*(1), 119-139. doi:10.1006/jcss.1997.1504

Gough, B. (2005). *An introduction to GCC* (1 ed.). Bristol: Network theory.

IEEE Standard Glossary of Software Engineering Terminology. (1990).

Indika. (2011). *Difference Between Strong AI and Weak AI*. Recuperado el 22 de 04 de 2017, de Differencebetween.com: http://www.differencebetween.com/difference-between-strong-ai-and-vs-weak-ai/

Joachims, T. (2007). Search Engines that Learn from Implicit Feedback. *Computer, 40*(8), 34-40.

Joachims, T. (2013). *Learning to Classify Text Using Support Vector Machines* (1 ed.). Springer-Verlag New York Inc.

Lichman, M. (2013). *UCI Machine Learning Repository.* University of California, Irvine, School of Information and Computer Sciences. Obtenido de http://archive.ics.uci.edu/ml

Martínez-Muñoz, G., & Suárez, A. (2005). Switching class labels to generate classification ensembles. *Pattern Recognition, 38*(10), 1483–1494.

Prokhorov, D. (2001). *IJCNN 2001 neural network competition.* Slide presentation in IJCNN'01, Ford Research Laboratory. Obtenido de http://www.geocities.com/ijcnn/nnc_ijcnn01.pdf

Russell, S., & Norvig, P. (2016). *Artificial intelligence* (1 ed.). Boston: Pearson.

Schneider, J. (1997). *Cross Validation.* Recuperado el 09 de 02 de 2016, de https://www.cs.cmu.edu/~schneide/tut5/node42.html

Sierra, B. (2006). *Aprendizaje automático* (1 ed.). Madrid: Pearson Prentice Hall.

Sommerville, I. (2005). *Ingeniería del software* (7 ed.). Pearson.

Anexos

A. Tipos de Núcleos

Las máquinas de soporte vectorial pueden utilizar diferentes núcleos para buscar un hiperplano separador en espacios transformados de múltiples dimensiones.

Los núcleos más utilizados son los siguientes:

- **Núcleo Lineal**

  El núcleo lineal permite separar los problemas linealmente. Su función es la siguiente:

$$K(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^{\mathrm{T}} * \boldsymbol{v}$$

  Dónde $\boldsymbol{u}$ y $\boldsymbol{v}$ son dos vectores de ejemplos.

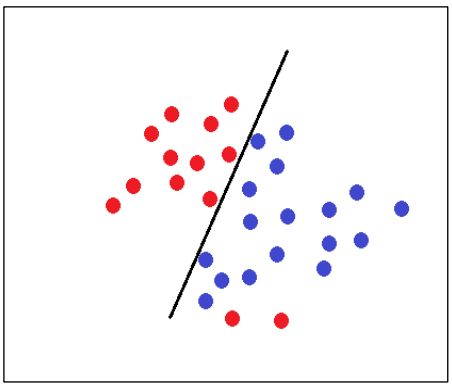

Figura 24. Núcleo Lineal

- **Núcleo Polinomial**

  El núcleo polinomial permite separar problemas no separables linealmente con una función polinómica. Los hiperparámetros que utiliza el núcleo polinomial, son la pendiente γ, la constante polinomial $coef$ y el grado polinomial n.

$$K(\boldsymbol{u}, \boldsymbol{v}) = (\gamma * \boldsymbol{u}^{T} * \boldsymbol{v} + coef)^{n}$$

  Dónde $\boldsymbol{u}$ y $\boldsymbol{v}$ son dos vectores de ejemplos.

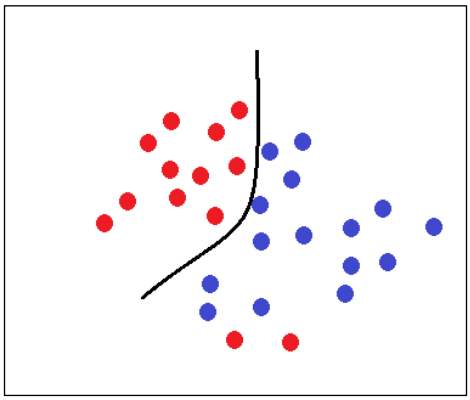

Figura 25. Núcleo Polinomial

- **Núcleo RBF**

  El núcleo gaussiano RBF se utiliza para problemas no separables linealmente. Separa el problema utilizando la función *Base Radial*. El hiperparámetro que se utiliza es la anchura del núcleo γ.

  La anchura del núcleo (γ), es muy importante definirlo correctamente. Ya que si γ es demasiado grande tiende a que el problema se sobreajuste. Si γ es demasiado pequeño se consigue un modelo muy restringido. Esto hará, que el núcleo se comporte de manera similar que un núcleo lineal.

$$K(\boldsymbol{u}, \boldsymbol{v}) = exp(-\gamma||\boldsymbol{u} - \boldsymbol{v}||^2)$$

  Dónde $\boldsymbol{u}$ y $\boldsymbol{v}$ son dos vectores de ejemplos.

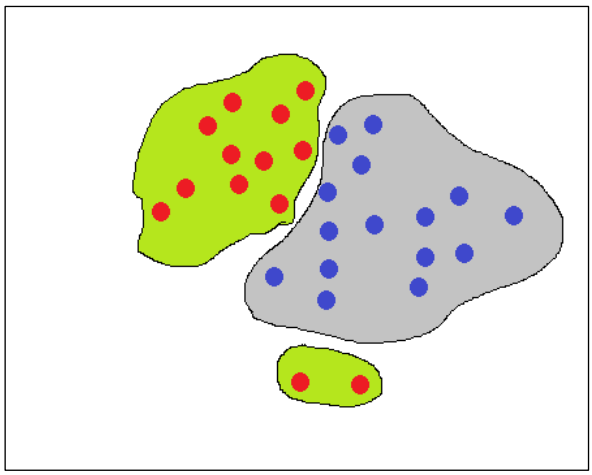

Figura 26. Núcleo RBF

B. Licencias de software libre

En este apartado, se van a explicar las licencias de software libre más importantes:

- **Licencia BSD (*Berkeley Software Distribution*)**: Es una licencia de software libre que permite el uso del código fuente y modificarlo. También se puede realizar un software privado. Existen 3 tipos de licencia BSD, de 2, 3 y 4 clausulas.
    - **Licencia BSD original (4 clausulas)**:

      1. Las redistribuciones del código fuente deben conservar el aviso de copyright anterior, esta lista de condiciones y el siguiente descargo de responsabilidad.

      2. Las redistribuciones en formato binario deben reproducir el aviso de copyright anterior, esta lista de condiciones y el siguiente descargo de responsabilidad en la documentación y/u otros materiales suministrados con la distribución.

      3. Todo el material publicitario que mencione las funciones o utilice este software debe mostrar el siguiente reconocimiento: Este producto incluye software desarrollado por la Universidad de California, Berkeley y sus colaboradores.

      4. Ni el nombre de la Universidad ni los nombres de sus colaboradores pueden usarse para apoyar o promocionar productos derivados de este software sin permiso previo y por escrito.

    - **Licencia BSD modificada (3 cláusulas)**:

      1. Las redistribuciones del código fuente deben conservar el aviso de copyright anterior, esta lista de condiciones y el siguiente descargo de responsabilidad.

      2. Las redistribuciones en formato binario deben reproducir el aviso de copyright anterior, esta lista de condiciones y el siguiente descargo de responsabilidad en la documentación y/u otros materiales suministrados con la distribución.

      3. Ni el nombre de los titulares de derechos de autor ni los nombres de sus colaboradores pueden usarse para apoyar o promocionar productos derivados de este software sin permiso específico previo y por escrito.

    - **Licencia BSD modificada (2 clausulas)**:

      1. Las redistribuciones del código fuente deben conservar el aviso de copyright anterior, esta lista de condiciones y el siguiente descargo de responsabilidad.

      2. Las redistribuciones en formato binario deben reproducir el aviso de copyright anterior, esta lista de condiciones y el siguiente descargo de responsabilidad en la documentación y/u otros materiales suministrados con la distribución.

- **Licencia GPU GPL**: Es una licencia para software libre, que permite utilizar y modificar un software siempre que el nuevo software se cree bajo una licencia similar (*copyleft*). Con lo cual este tipo de licencia no te permite crear un software privado. Con esta licencia, se conservan los derechos de autor del desarrollador.
- **Licencia LGPL**: Esta licencia es muy parecida a la licencia GPU GPL, con la diferencia de que el nuevo software con esta licencia, permite integrarse en softwares privados.
- **Licencia AGPL**: Esta licencia, es similar a la licencia GPU GPL. Pero recomendada para que se ejecute el software de manera habitual en la red.
- **Licencia Apache**. La licencia Apache, permite al usuario a modificar y distribuir versiones modificadas del software. Esta licencia, no exige que los nuevos softwares desarrollados usen la misma licencia, ni que tengan que seguir siendo de uso abierto.

C. Parámetros de entrada de entrada de la ampliación de la librería EnsembleSVM

El programa utiliza una serie de hiperparámetros y parámetros de entrada, para configurar el problema antes de obtener el predictor. Estos hiperparámetros y parámetros son los siguientes:

- **-c**: Valor de la regularización
- **-gamma**: Valor de la anchura del núcleo RBF
- **-data**: Nombre del fichero de entrada
- **-o**: Nombre del fichero de salida
- **-labels**: Literal de las dos etiquetas del conjunto de ejemplos de entrenamiento
- **-bootstrap**: Nombre del fichero que contiene los conjuntos creados con *bootstrap*
- **-classSwitching**: Nombre del fichero que contiene los conjuntos creados con *class-switching*
- **-nmodels**: Número de predictores que quieren generar
- **-nboot**: Número de conjuntos que se van a crear con *bagging* para obtener los predictores, tiene que ser el mismo valor que nmodels
- **-nclassSwitching**: Número de conjuntos que se van a crear con *class-switching* para obtener los predictores, tiene que ser el mismo valor que nmodels
- **-pclassSwitching:** Porcentaje del total de ejemplos a cambiar la clase.
- **-pRem:** Porcentaje de remuestreo que se quiere utilizar en el problema

Como salida del programa se genera un fichero con los predictores entrenados. Este fichero almacena el tipo de máquina de soporte vectorial que se ha utilizado, el tipo de núcleo, el número de predictores que se han creado, los valores de los hiperparámetros C y γ utilizados, y los vectores soporte de cada predictor. El fichero que se genera también se puede utilizar como entrada del programa. De este modo, se puede predecir las etiquetas de un fichero test, sin tener que volver a realizar el entrenamiento de los predictores.

D. Tipo de ficheros de entrada y salida

Los ejemplos etiquetados, se introducen en el sistema mediante un fichero de datos. Los datos del fichero, tienen que estar en un determinado formato, para que el programa no de error. Los formatos que soporta el programa son: Dense CSV, Sparse CSV y Por defecto (Claesen, 2014).

- Dense CSV: La estructura de este formato para los ejemplos etiquetados, se forma por la etiqueta, separado por el delimitador ",", y las características del ejemplo separados por el mismo delimitador. Todos los ejemplos tienen que estar formados por el mismo número de características. Por ejemplo, en un fichero con formato Dense CSV con 5 características. Las características, pueden ser de cualquier valor, ya sea numérico o de caracteres. Para los ejemplos se van a utilizar números enteros:

  POSITIVO,1,4,0,2,3

  NEGATIVO,2,4,1,0,3

  POSITIVO,0,3,2,4,0

- Sparse CSV: Este formato es parecido al Dense CSV, se diferencia en que las características se almacenan de diferente manera. Los ejemplos están formados por el número de columna al que pertenece y el valor de la característica, separados por el delimitador ":". En este formato cuando una característica es de valor cero, la omite. Por ejemplo, un fichero con formato Sparse CSV con 5 características:

  POSITIVO,1:1,2:4,4:2,5:3

  NEGATIVO,1:2,2:4,3:1,5:3

  POSITIVO,2:3,3:2,4:4

- Por defecto: El formato Por defecto es igual que el Sparse CSV, pero sin delimitadores. Por ejemplo, un fichero con 5 características:

  POSITIVO 1:1 2:4 4:2 5:3

  NEGATIVO 1:2 2:4 3:1 5:3

  POSITIVO 2:3 3:2 4:4

El programa cuando realiza una predicción de un fichero de ejemplos, genera un fichero de salida. El fichero de salida, recoge por cada ejemplo, la etiqueta que se ha obtenido al utilizar voto por mayoría, y el valor decimal con el que se ha predicho dicha etiqueta. Por ejemplo, tenemos tres ejemplos a etiquetar con los predictores obtenidos a partir del conjunto de ejemplos etiquetados del punto anterior:

Ejemplo 1 1:1 2:4 4:2 5:3

Ejemplo 2 1:2 2:4 3:1 5:3

Ejemplo 3 2:3 3:2 4:4

El fichero de salida generado es el siguiente:

1 0.87

-1 0.56

1 0.67

En el programa se indica como parámetro de entrada las etiquetas clase que tiene el conjunto de entrenamiento. A continuación, el programa renombra las etiquetas clase con 1 y -1. Por ejemplo, en los ejemplos anteriores la etiqueta POSITIVO la renombra con 1 y la etiqueta NEGATIVO con -1.